

Cadence® Virtual Component Co-Design Architecture Services Tutorial

**Product Version 2.1
March 2001**

© 2001 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 1-800-862-4522.

All other trademarks are the property of their respective holders.

Restricted Print Permission: This publication is protected by copyright and any unauthorized use of this publication may violate copyright, trademark, and other laws. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. This statement grants you permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used solely for personal, informational, and noncommercial purposes;
2. The publication may not be modified in any way;
3. Any copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement; and
4. Cadence reserves the right to revoke this authorization at any time, and any such use shall be discontinued immediately upon written notice from Cadence.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

1

<u>Architecture Services Tutorial</u>	5
<u>Start the VCC Software</u>	6
<u>Create a Behavior Diagram</u>	7
<u>Instantiate the Behavior Blocks</u>	7
<u>Save the Behavior Diagram</u>	13
<u>Specify Parameter Values</u>	15
<u>Wire the Blocks</u>	16
<u>Generate a Netlist</u>	17
<u>Create an Architecture Diagram</u>	18
<u>Instantiate the Architecture Blocks</u>	18
<u>Specify Bus Properties</u>	20
<u>Wire the Architecture Blocks</u>	22
<u>Bind the Views</u>	27
<u>Service Declarations</u>	28
<u>Architecture Services</u>	30
<u>Specify Parameter Values</u>	33
<u>Generate a Netlist</u>	38
<u>Create a Mapping Diagram</u>	39
<u>Instantiate Behavior and Architecture Blocks</u>	40
<u>Add Mapping Connections</u>	41
<u>Add Performance Views</u>	43
<u>Specify a Memory Segment Parameter</u>	46
<u>Add a Mapping Pattern</u>	48
<u>Generate a Netlist</u>	50
<u>Run a Simulation</u>	51
<u>Supplemental Exercise:</u>	
<u>Add Memory Reference Calls to a Blackbox Model</u>	54
<u>Specify the Performance Model</u>	58
<u>Embed Delays</u>	60

VCC Architecture Services Tutorial

Supplemental Exercise:

<u>Create a Custom Architecture Model</u>	64
<u>Create a New Block</u>	64
<u>Add Service Declarations</u>	66
<u>Bind the View</u>	67

<u>Glossary</u>	69
-----------------------	----

Architecture Services Tutorial

Architecture services are C++ models for analyzing the performance of architecture resources and communication patterns.

In this tutorial, you will perform the following exercises:

- Create a Behavior Diagram
- Create an Architecture Diagram
- Create a Mapping Diagram
- Run a Simulation

In addition, you can perform the following supplemental exercises:

- Add Memory Reference Calls to a Blackbox Model
- Create a Custom Architecture Model

Important

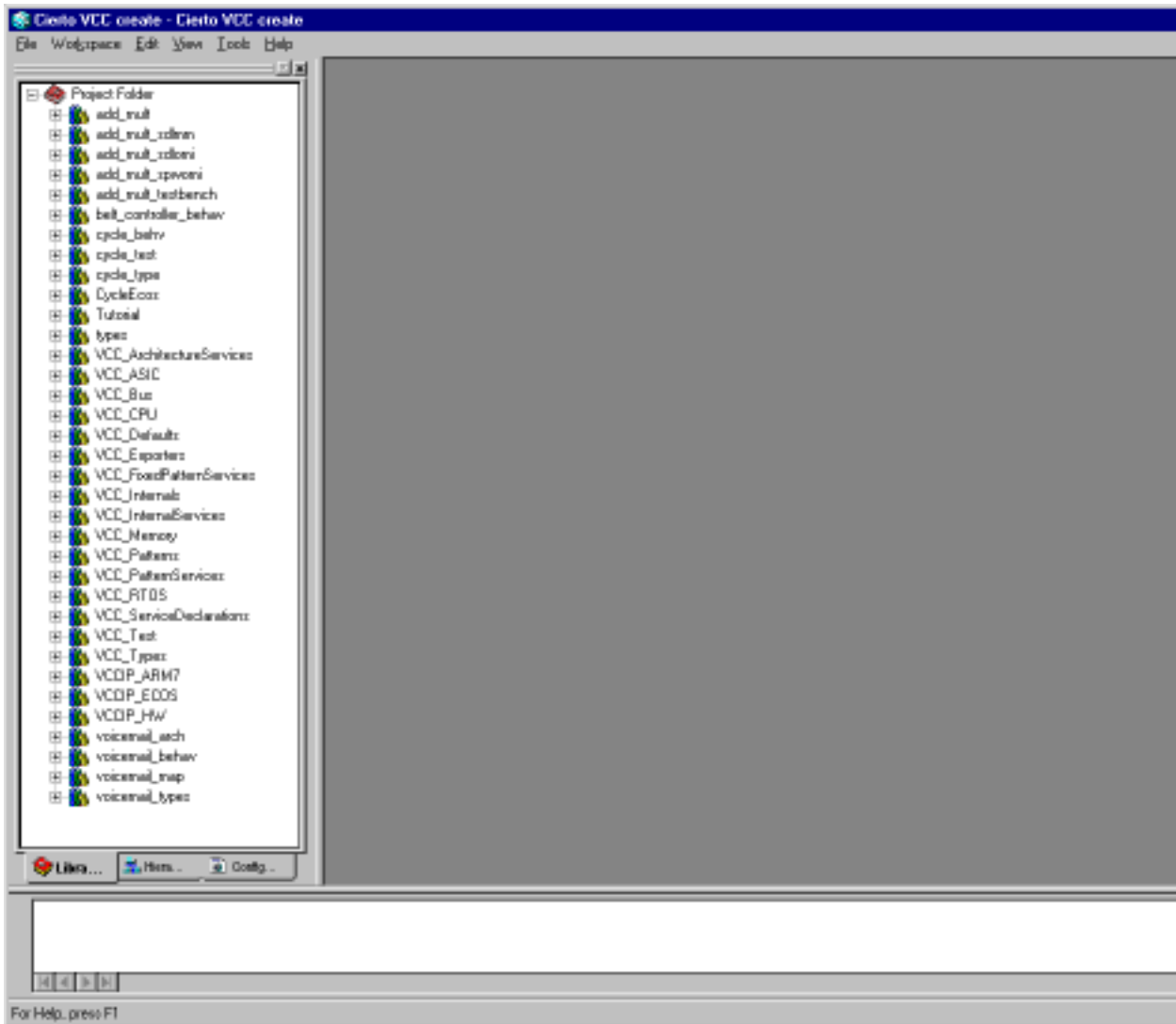
To provide optimum screen space for the Cadence® Virtual Component Co-Design (VCC) tool, you should use the tutorial in printed form. Print the PDF version of this tutorial before you begin the exercises.

Start the VCC Software

1. To start the VCC Create software, choose *Start > Programs > Ciertto VCC > Create*.
2. If VCC Create opens with all the toolbars expanded, use the *View > Toolbars* menu command to close them:

- ☐ Choose *View > Toolbars > <Toolbar name>*.
- ☐ One at a time, choose each of the toolbars with a check in front of it.

Closing the toolbars gives you more screen space for your designs.

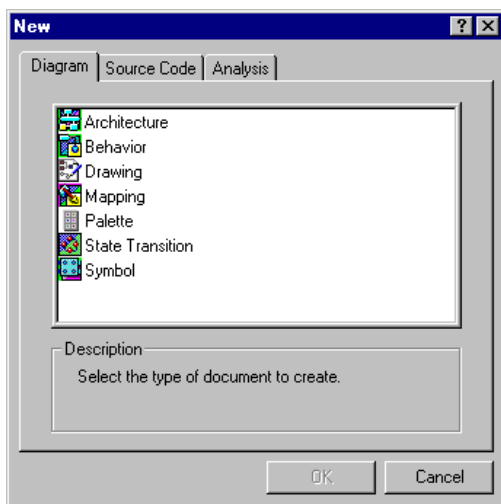


Create a Behavior Diagram

A behavior model defines the functionality used in a simulation. The behavior diagram you will create contains instances of behavior primitives, parameters assigned to those primitives, and wires connecting the primitive models.

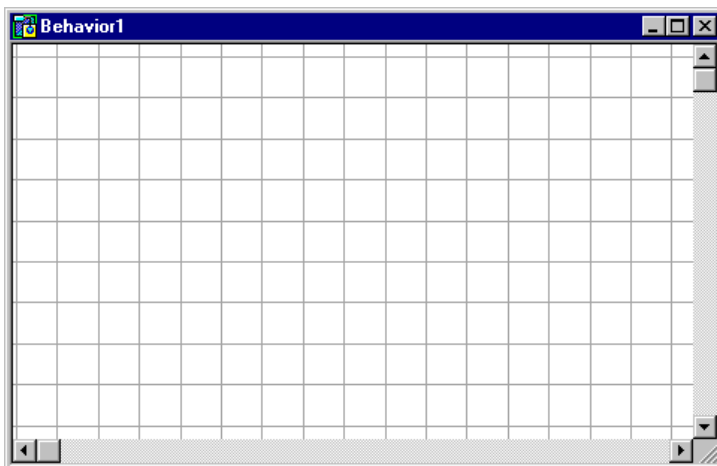
Instantiate the Behavior Blocks

1. Choose *File > New*.



2. Double-click the *Behavior* icon.

A new empty design titled *Behavior1* opens.

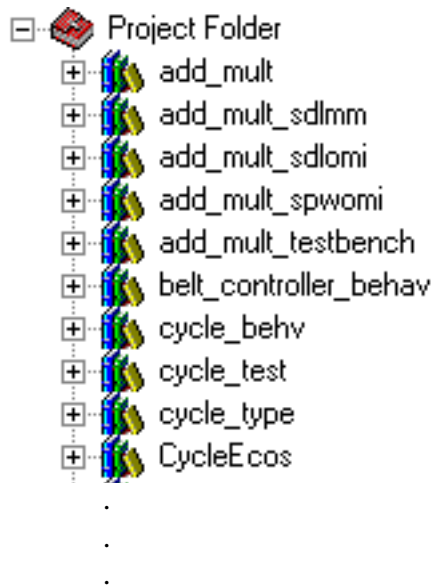


Note: To toggle the grid on and off, choose *View > Grid*.

VCC Architecture Services Tutorial

Architecture Services Tutorial

On the left side of the Create window is the expanded Project Folder that contains all the libraries you can use (these are defined in your *cds.lib* file).

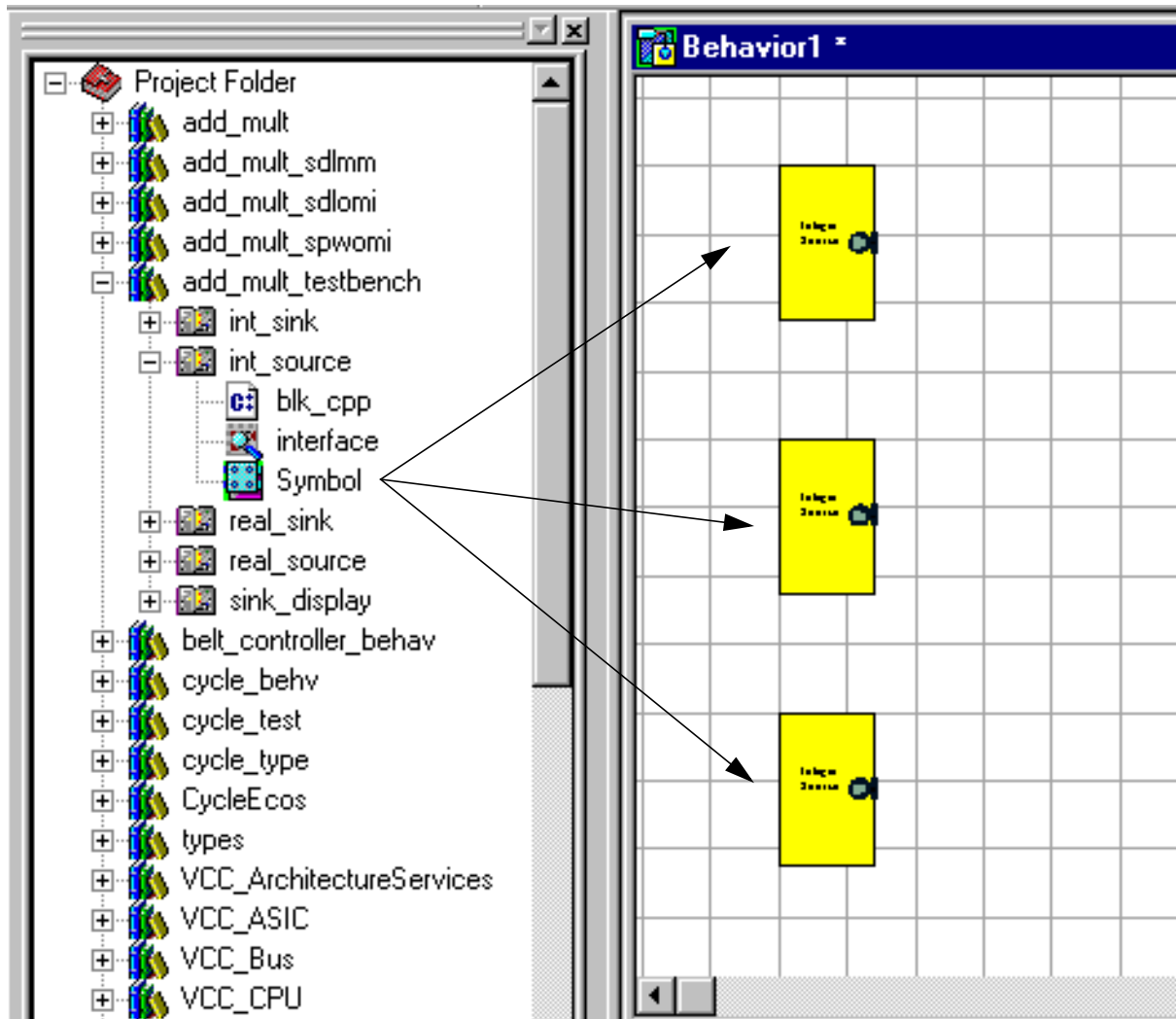


VCC Architecture Services Tutorial

Architecture Services Tutorial

3. In the Project Folder, expand the *add_mult_testbench* library, then expand the *int_source* cell.
4. Instantiate three copies of the *int_source* symbol view in your design.

To instantiate a copy, click on the symbol view, drag the symbol to your design, and drop the symbol.



5. When you finish instantiating these blocks, click the minus sign to collapse the expanded *int_source* cell.

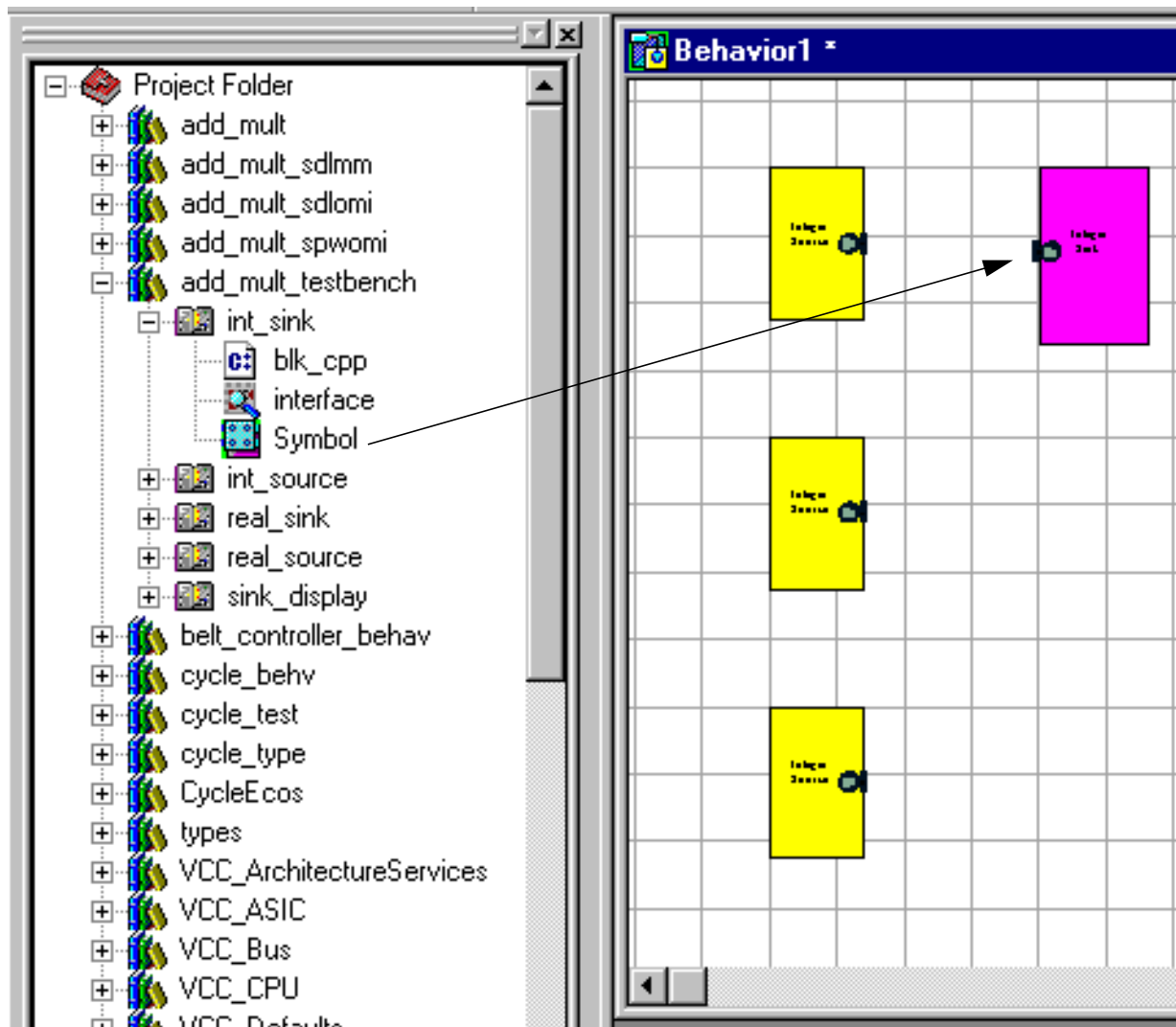
Leave the *add_mult_testbench* library expanded.

Note: If you don't collapse each cell after you use it, your Project Folder will grow increasingly long, making it hard to locate each subsequent cell in this tutorial.

VCC Architecture Services Tutorial

Architecture Services Tutorial

6. In the Project Folder, expand the *int_sink* cell.
7. Instantiate a copy of the *int_sink* symbol view in your design.

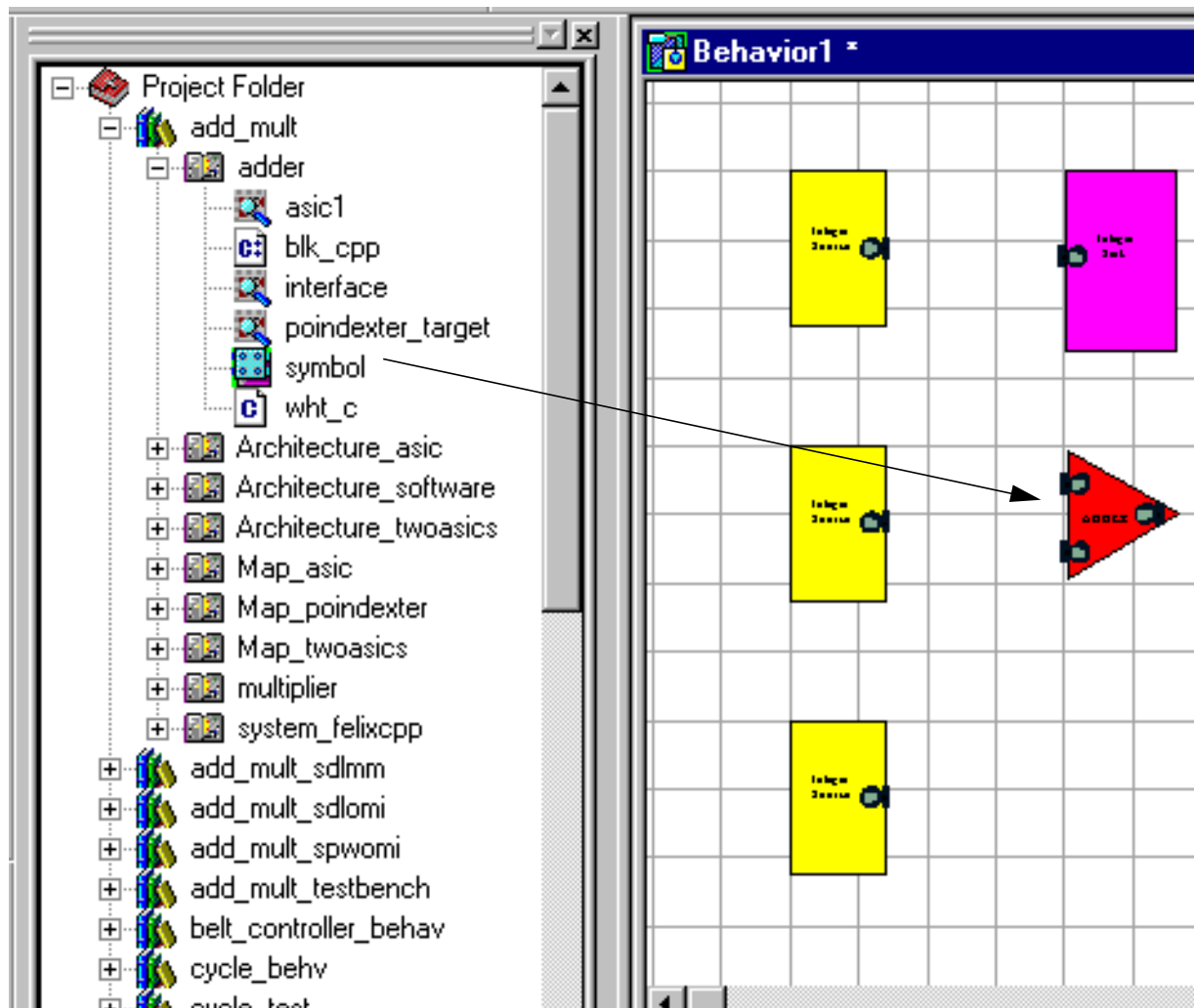


8. When you finish instantiating this block, click the minus signs to collapse the expanded *int_sink* cell and the *add_mult_testbench* library.

VCC Architecture Services Tutorial

Architecture Services Tutorial

9. In the Project Folder, expand the *add_mult* library, then expand the *adder* cell.
10. Instantiate a copy of the *adder* symbol view in your design.



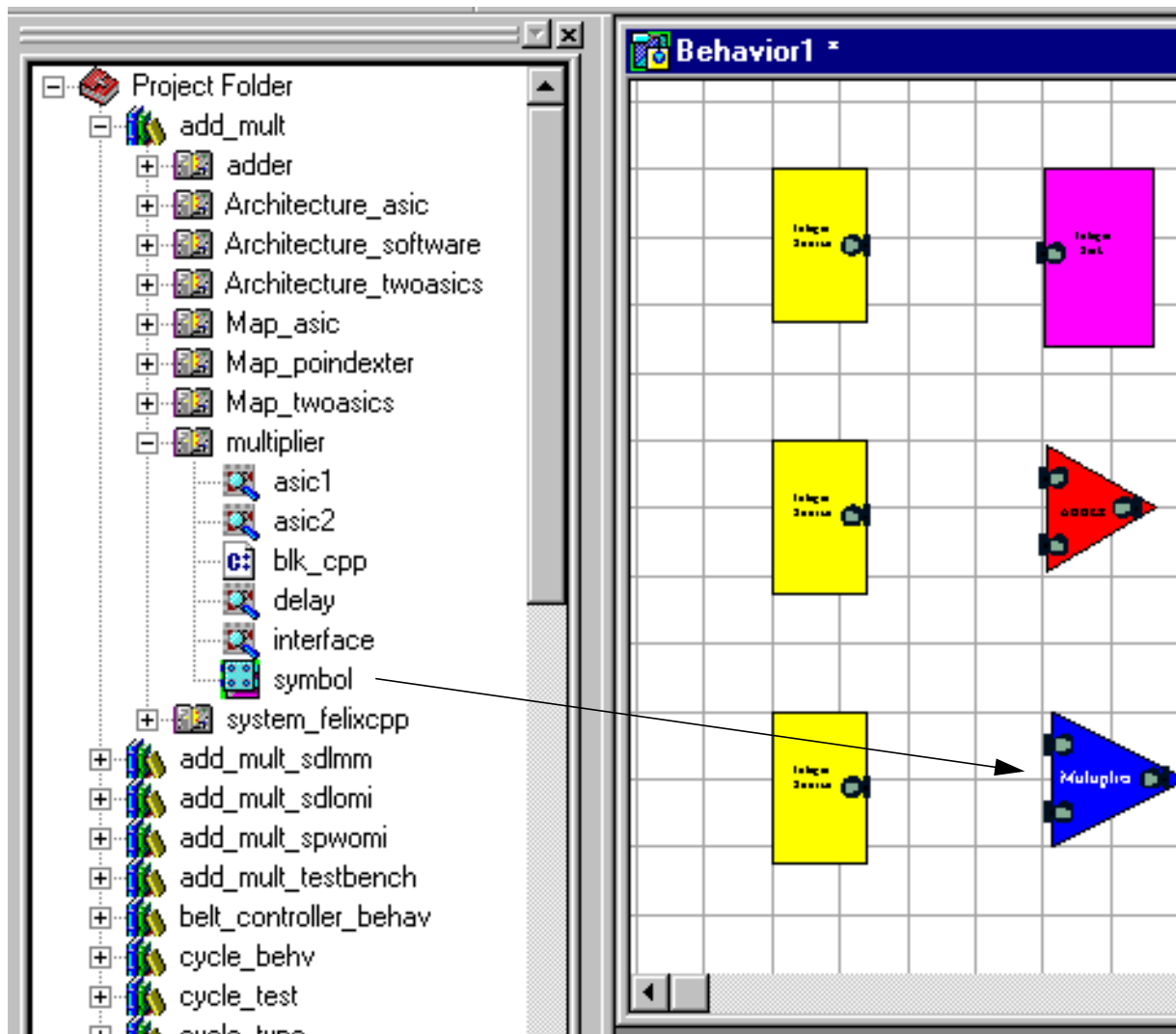
11. When you finish instantiating this block, click the minus signs to collapse the expanded *adder* cell.

Leave the *add_mult* library expanded.

VCC Architecture Services Tutorial

Architecture Services Tutorial

12. In the Project Folder, expand the *multiplier* cell in the *add_mult* library.
13. Instantiate a copy of the *multiplier* symbol view in your design.



14. When you finish instantiating this block, click the minus signs to collapse the expanded *multiplier* cell and the *add_mult* library.

Save the Behavior Diagram

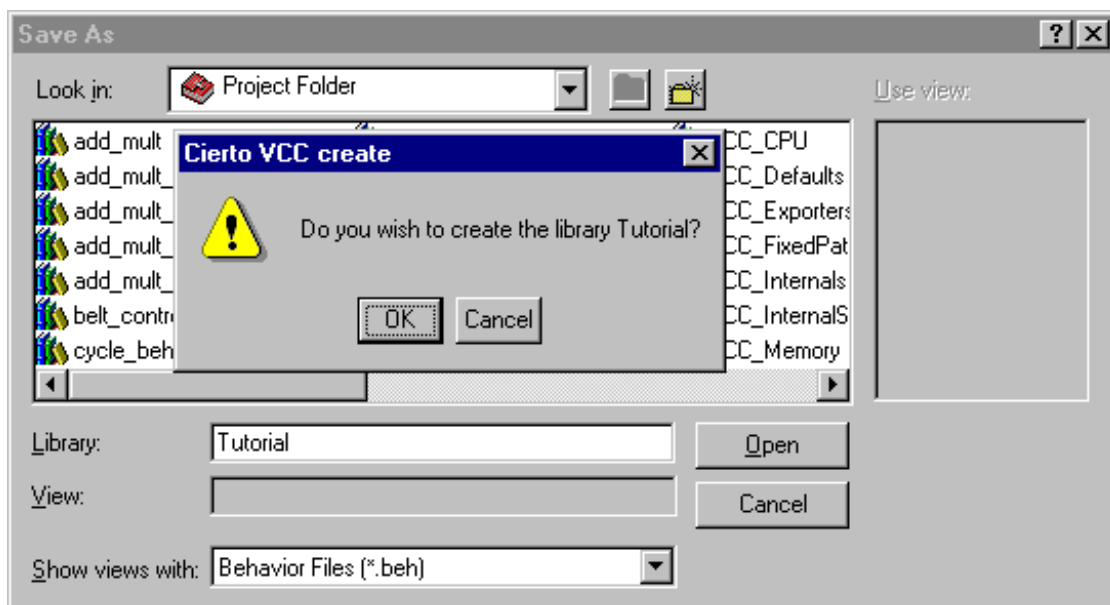
When you save a diagram for the first time, you must specify a *library.cell:view* for the design.

1. To save the behavior diagram, choose *File > Save*.

VCC prompts you to choose a library.

2. To create a new tutorial library, enter the name *Tutorial* in the Library field, then click *Open*.

VCC prompts you to create the new Tutorial library.



3. Click *OK* in the dialog.

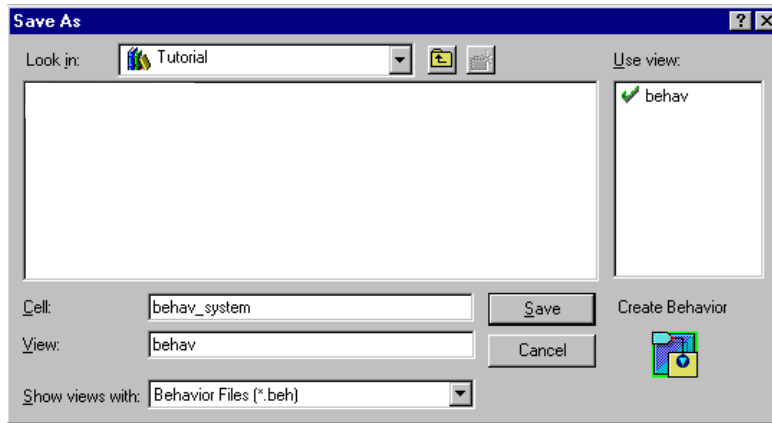
VCC creates the new Tutorial library and prompts you for a cell name.

VCC Architecture Services Tutorial

Architecture Services Tutorial

4. Enter the name *behav_system* and click *Save*.

The view name defaults to *behav*.



VCC adds your new library and cell to the library list.



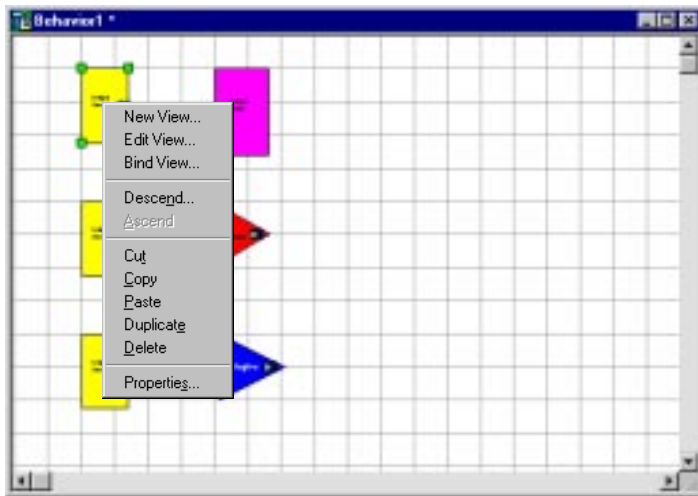
Note: If you were unable to complete the steps to instantiate the blocks:

- ☐ Close your *Tutorial.behav_system:behav* cellview
- ☐ Open *Tutorial_Examples.Example1:behav*
- ☐ Save the example as *Tutorial.behav_system:behav*

Specify Parameter Values

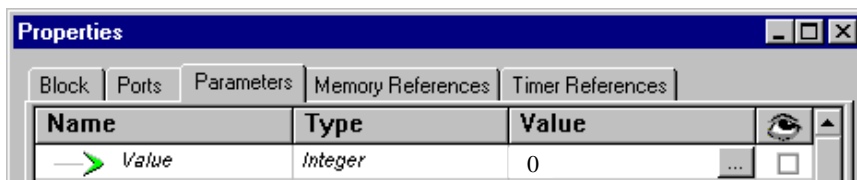
The inputs for this behavior model come from the three integer source blocks. You will assign each of them an integer value of 1, 2, and 3, respectively.

1. Right click on the first *Integer Source* symbol and choose *Properties* from the pop-up menu.



The Properties dialog opens.

2. Click the *Parameters* tab on the Properties dialog.



3. Double-click in the Value field, then replace the zero with the integer 1.
4. Click *OK* in the Properties dialog.
5. Repeat steps 1 through 4 for the other two integer source blocks, setting the value to 2 for the second block and 3 for the third block.

Note: If you were unable to complete the steps to set the parameter values:

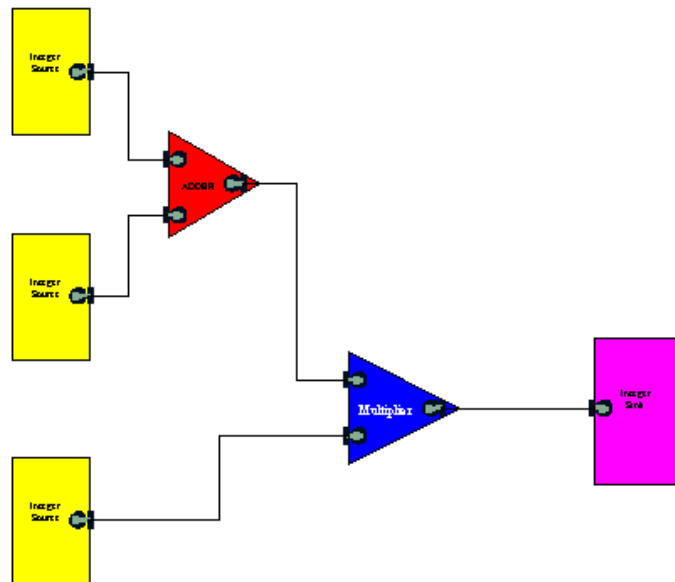
- ☐ Close the *Tutorial.behav_system:behav* cellview
- ☐ Open *Tutorial_Examples.Example2:behav*
- ☐ Save the example as *Tutorial.behav_system:behav*

Wire the Blocks

Connection wires connect outputs to inputs in the behavior diagram. You will wire the blocks in the behavior diagram in the following way:

- ☐ The outputs of the first two *Integer Sources* are inputs for the *Adder*. (Add 1 + 2 to get 3.)
- ☐ The output of the *Adder* and the output of the third *Integer Source* are inputs for the *Multiplier*. (Multiply 3 x 3 to get 9.)
- ☐ The output of the *Multiplier* is the input for the *Integer Sink*. (Discard the 9.)

1. Rearrange the blocks in this general order:



2. To add a wire, choose *Behavior > Wire*.
3. Click at the starting point for the wire (an output), then click at the ending point (an input) to terminate the wire.

VCC automatically inserts bends in the wire; you need to enter only the beginning and ending points.

Note: After you have wired a block, you can move the block and the wires move with it.

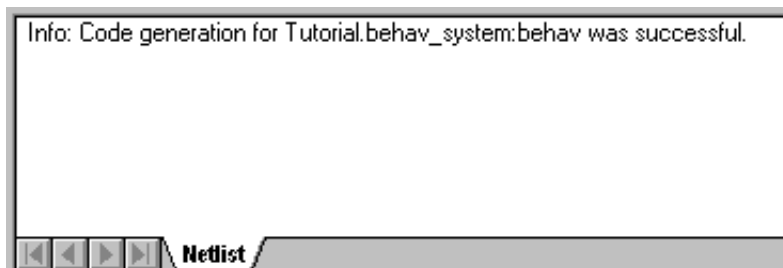
4. Wire each of the blocks as shown in this diagram.

Generate a Netlist

The Generate Netlist function verifies the syntax of your code and reports any errors.

1. To generate and verify the code for this diagram, choose *Tools > Generate Netlist*.

VCC reports the success of the code generation in the text field at the bottom of the VCC Create window:



If the code generation was unsuccessful, the messages should help you understand what went wrong. For example, the message might tell you that your design has a parameter with no value.

```
Error: Parameter 'Value' has no value.  
{in /(Tutorial.behav_system:behav)I_O}:
```

2. If VCC reports any errors, correct them and regenerate the netlist until all errors are gone before proceeding with the exercises in this tutorial.

Note: If you are unable to successfully generate the netlist:

- ☐ Close your *Tutorial.behav_system:behav* cellview
- ☐ Open *Tutorial_Examples.Example3:behav*
- ☐ Save the example as *Tutorial.behav_system:behav*

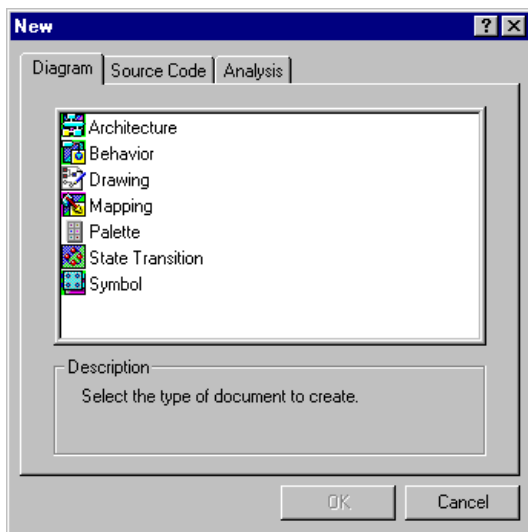
3. To save the behavior model, choose *File > Save*.
4. To close the behavior model, choose *File > Close*.

Create an Architecture Diagram

An architecture model defines the resources used in a simulation. The architecture diagram you will create contains instances of architecture primitives, parameters assigned to those primitives, and buses connecting the primitive models.

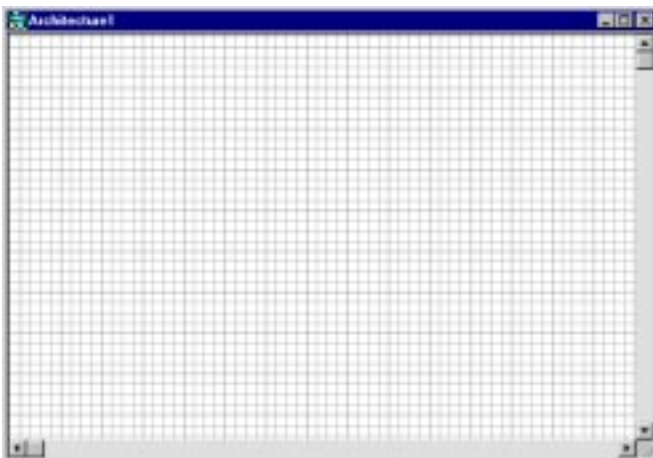
Instantiate the Architecture Blocks

1. Choose *File > New*.



2. Double-click the *Architecture* icon.

A new empty design titled *Architecture1* opens.



Note: To toggle the grid on and off, choose *View > Grid*.

VCC Architecture Services Tutorial

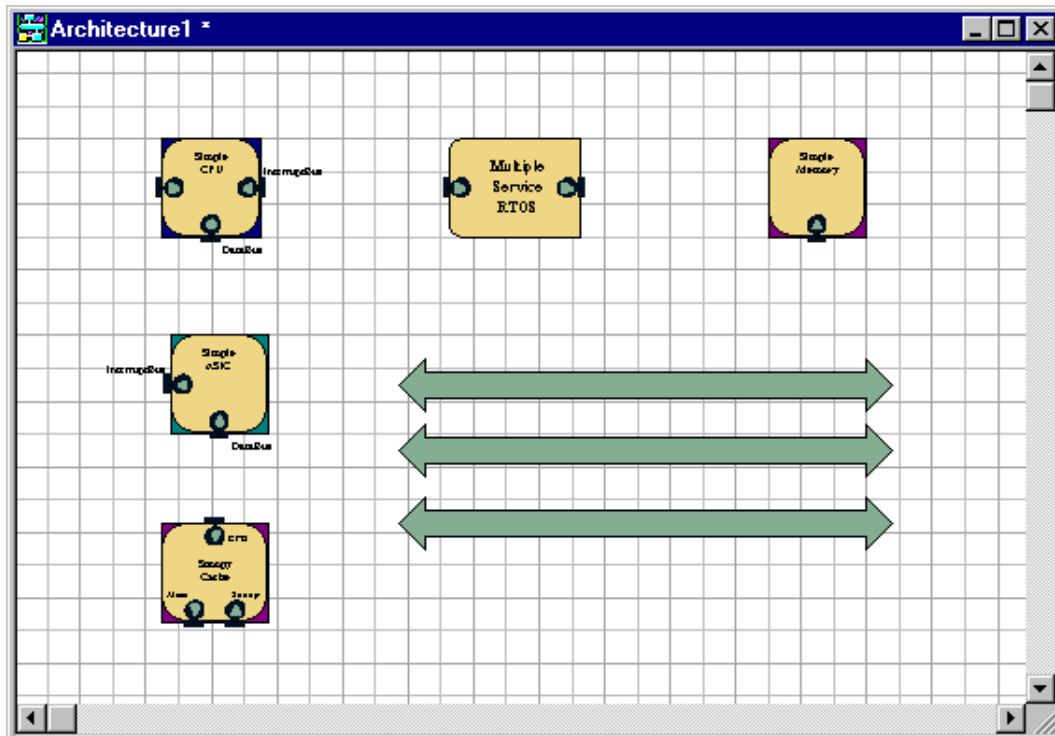
Architecture Services Tutorial

3. In the Project Folder, expand the *VCC_RTOS* library, then expand the *MultipleServiceRTOS* cell.
4. Instantiate the symbol view in your design.
5. Repeat steps 3 and 4 for the following lib.cell:views:
 - ☐ *VCC_CPU.SimpleCPU:symbol*
 - ☐ *VCC_ASIC.SimpleASIC:symbol*
 - ☐ *VCC_Memory.SnoopyCache:symbol*
 - ☐ *VCC_Memory.SimpleMemory:symbol*
6. To add a bus to the diagram:
 - ☐ Choose *Architecture > Bus*.
 - ☐ Click in your diagram to indicate the starting point of the bus, move your cursor, then double-click to terminate the bus.

The length of the bus doesn't matter; you'll modify it later.

7. Repeat step 6 until you have three buses in your diagram.

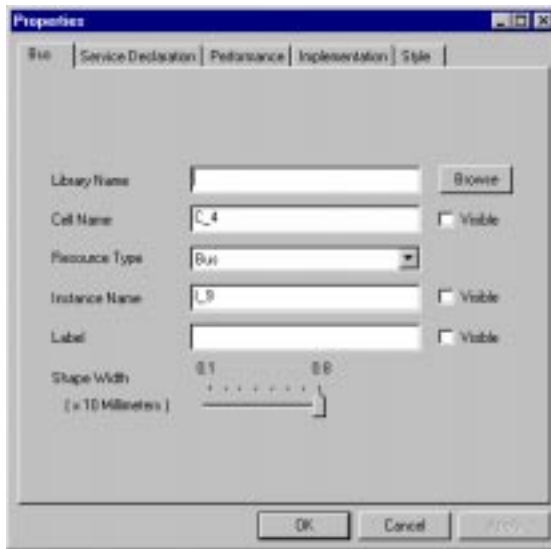
Your diagram should contain these instances:



Specify Bus Properties

1. To specify properties for each of the buses, right click on the bus and choose *Properties* from the pop-up menu.

The Properties dialog prompts you for the Library Name and Cell Name of the bus.



You will specify two of these buses as *FCFSBus* and one as an *InterruptBus*.

2. Click *Browse* and open the *VCC_Bus* library in the browser.
3. Click the *FCFSBus* cell name, then click *Open*.

The Properties dialog displays the library and cell names.



VCC Architecture Services Tutorial

Architecture Services Tutorial

4. Click *Visible* next to the Cell Name field.

This displays the *FCFSBus* cell name on the bus graphic in the diagram.

5. Click the Resource Type down arrow and choose *Data Bus*.
6. Click *OK* in the Properties dialog.
7. Repeat steps 1 through 6 for the second bus and configure it as an *FCFSBus* cell.
8. Repeat steps 1 through 6 for the third bus and configure it as an *InterruptBus* cell, specifying the Resource Type as *Bus*.

Now you are ready to wire the blocks and buses in your architecture diagram.

9. To save this diagram, choose *File > Save* and specify the library.cell:view as *Tutorial.arch_system:arch*.

Note: If you were unable to instantiate these architecture blocks and specify the bus properties:

- ☐ Close your *Tutorial.arch_system:arch* cellview
- ☐ Open *Tutorial_Examples.Example4:arch*
- ☐ Save the example as *Tutorial.arch_system:arch*

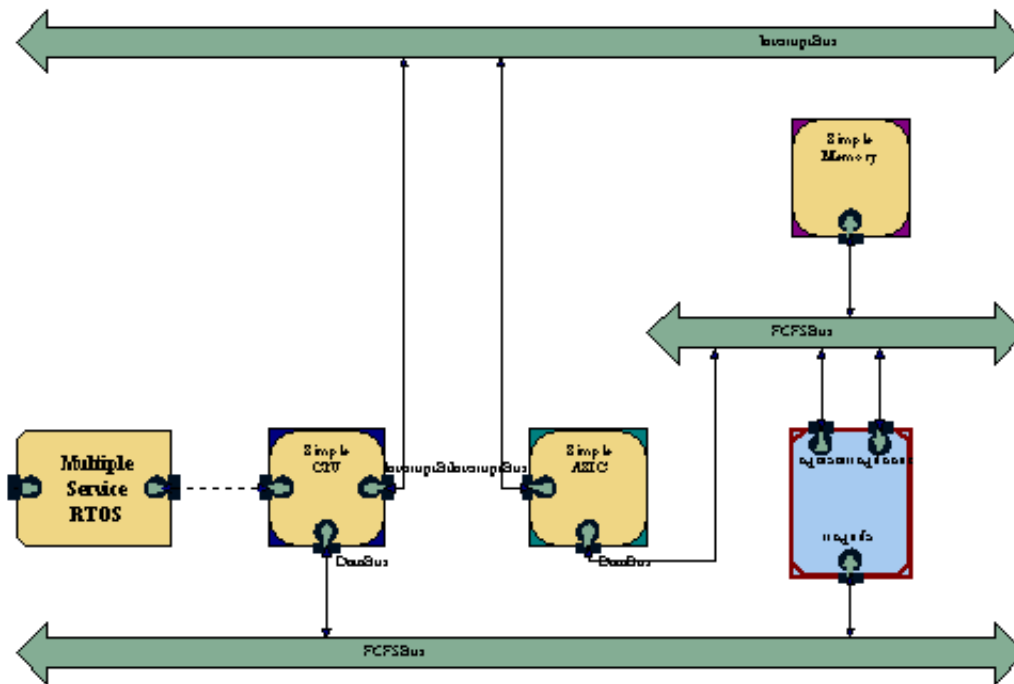
VCC Architecture Services Tutorial

Architecture Services Tutorial

Wire the Architecture Blocks

Architecture communication wires can connect blocks to blocks or blocks to buses.

When you complete the wiring steps in this section, your diagram should look like this:

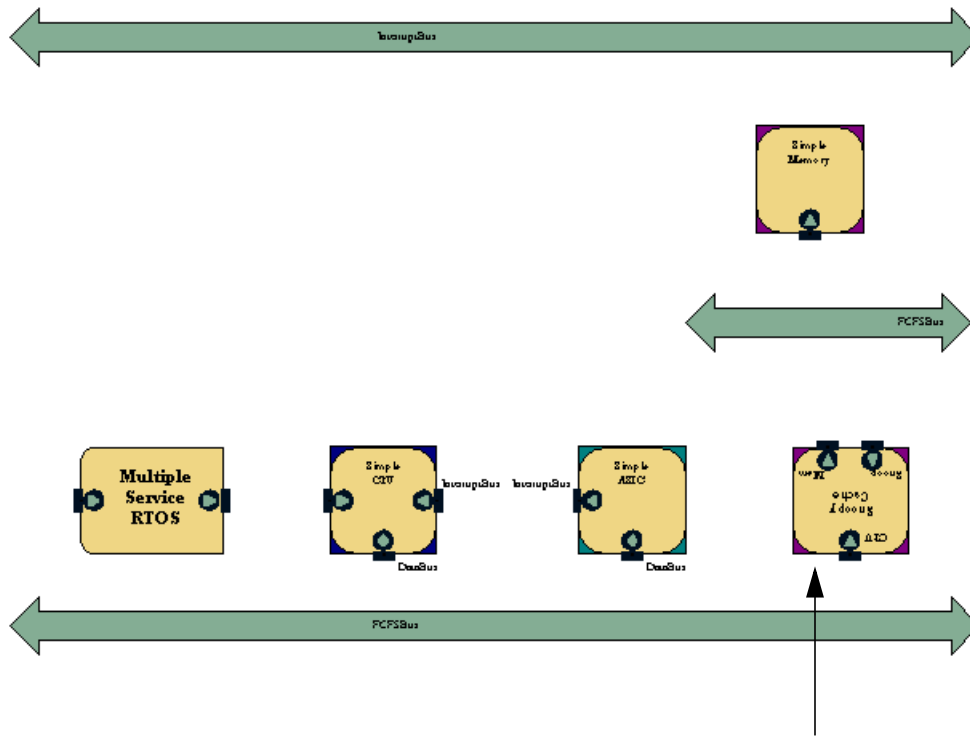


VCC Architecture Services Tutorial

Architecture Services Tutorial

Organize the Blocks

1. Arrange your blocks similar to this diagram:



Notice that the *SnoopyCache* block is flipped to make wiring easier.

2. To flip the *SnoopyCache* block:

- ☐ Click the block to select it.

Make sure you've selected the block and not a port. Bright green markers appear in all four corners of a block when it is selected.

- ☐ Choose *Layout > Flip Vertically*.

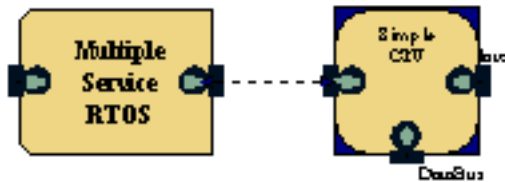
3. To stretch the buses, click on the end of the bus symbol and drag your cursor.

Wire the Blocks

1. To connect the *RTOS* to the *CPU*:

- ☐ Choose *Architecture > Scheduler Assignment*.
- ☐ Click the output port of the *RTOS*, then click on the input port of the *CPU*.

The connection looks like this:

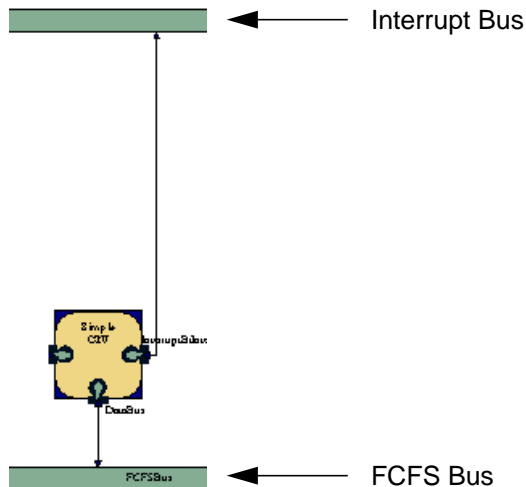


This connection is not an ordinary communication wire; it is a scheduler assignment. A scheduler represents a scheduling policy for an architecture resource that governs how the behavior models assigned to the resource gain access to and share the resource.

2. To connect the *CPU* to the *FCFS* and *Interrupt* buses:

- ☐ Choose *Architecture > Communication Wire*.
- ☐ Click the bottom output port of the *CPU*, then click on the edge of the *FCFS* bus below.
- ☐ Repeat this process, connecting the right output port to the *Interrupt* bus.

The connections look like this:



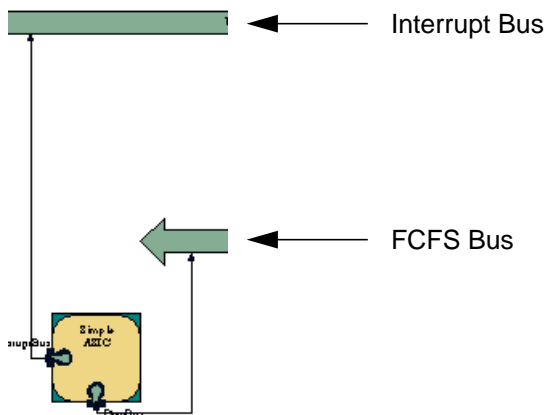
VCC Architecture Services Tutorial

Architecture Services Tutorial

3. To connect the *ASIC* to the *FCFS* and *Interrupt* buses:

- ❑ Choose *Architecture > Communication Wire*.
- ❑ Click the bottom output port of the *ASIC*, then click on the edge of the *FCFS* bus.
- ❑ Repeat this process, connecting the left output port to the *Interrupt* bus.

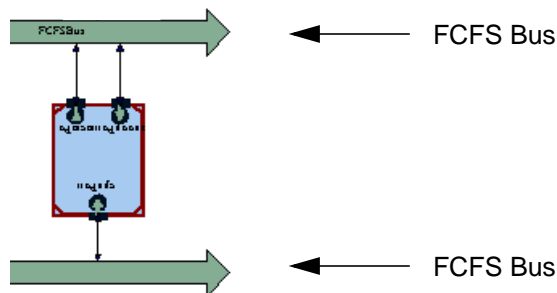
The connections look like this:



4. To connect the *SnoopyCache* to the *FCFS* bus:

- ❑ Choose *Architecture > Communication Wire*.
- ❑ Click either of the top output ports of the *SnoopyCache*, then click on the edge of the *FCFS* bus above the block.
- ❑ Repeat this process, connecting the other top output port to the same *FCFS* bus.
- ❑ Repeat this process, connecting the bottom output port to the *FCFS* bus below the block.

The connections look like this:



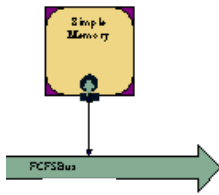
VCC Architecture Services Tutorial

Architecture Services Tutorial

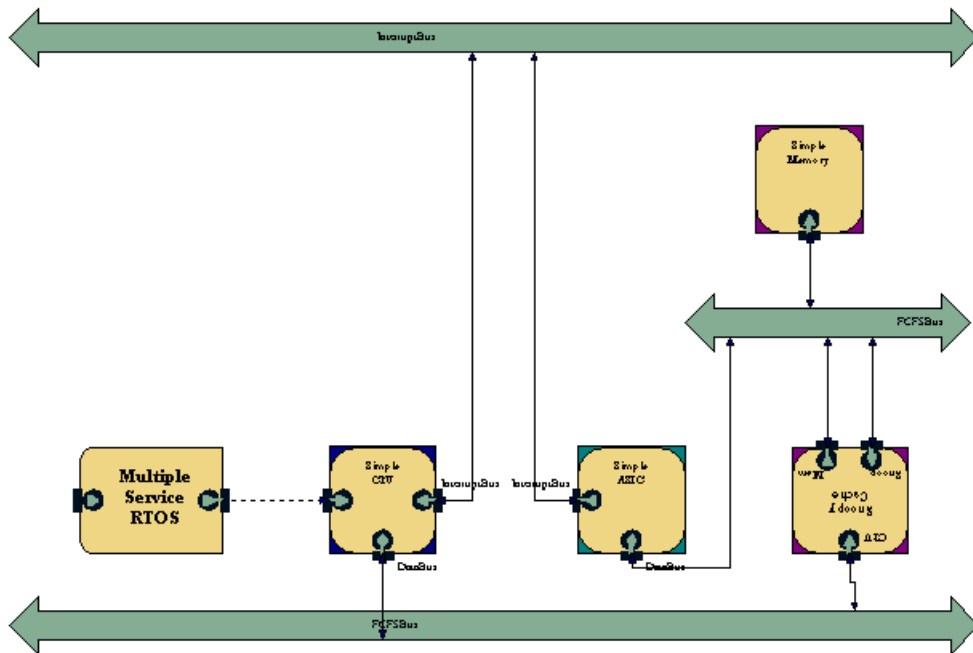
5. To connect the *SimpleMemory* to the *FCFS* bus:

- ❑ Choose *Architecture > Communication Wire*.
- ❑ Click the output port of the *SimpleMemory*, then click on the edge of the *FCFS* bus below the block.

The connection looks like this:



Your completed diagram with all the blocks and buses connected looks like this:



6. To save the *Tutorial.arch_system:arch* diagram, choose *File > Save*.

Note: If you were unable to wire these architecture blocks:

- ❑ Close your *Tutorial.arch_system:arch* cellview
- ❑ Open *Tutorial_Examples.Example5:arch*
- ❑ Save the example as *Tutorial.arch_system:arch*

Bind the Views

Performance analysis in the VCC environment requires a performance view for each architecture primitive. The performance view describes how the behavior that runs on the model affects the performance of the design.

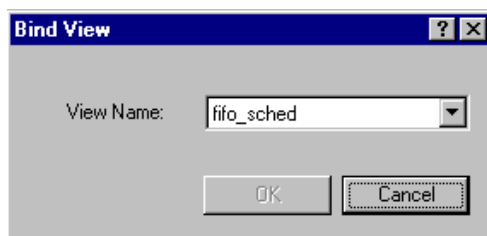
An architecture model can have more than one performance model. When you bind a performance view to an architecture model, you tell VCC which performance model to implement when it performs a simulation.

Each of the architecture blocks in the diagram is bound to a specific view:

MultipleServiceRTOS	<code>fifo_sched</code>
SimpleCPU	<code>processor</code>
SimpleASIC	<code>asic</code>
SnoopyCache	<code>cache</code>
SimpleMemory	<code>memory</code>
FCFSBus	<code>bus</code>
InterruptBus	<code>bus</code>

1. To open the Bind View dialog, right click on a block and choose *Bind View* from the pop-up menu.

The Bind View dialog lets you choose from a list of appropriate views. The choices are different for each type of architecture block.



2. To bind a view, click the down arrow, choose the appropriate view, then click *OK*.

Note: If the correct view is already specified, you can click *Cancel* in the Bind View dialog.

VCC Architecture Services Tutorial

Architecture Services Tutorial

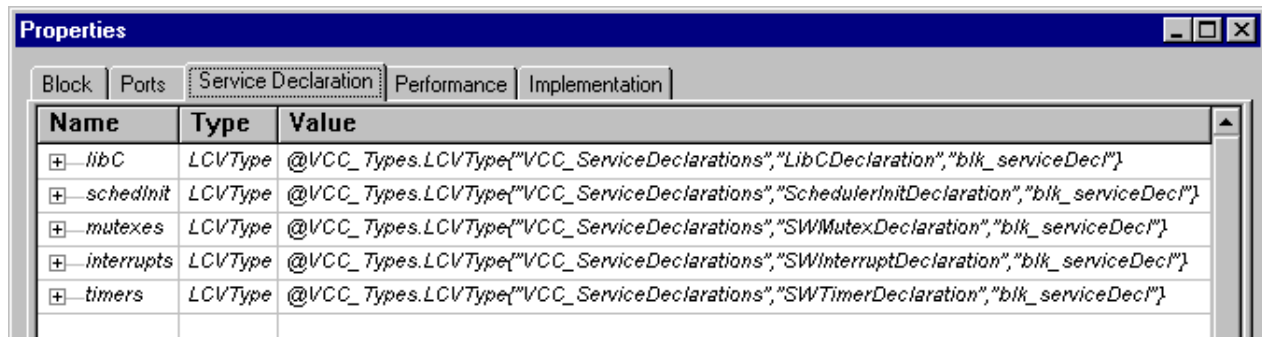
Service Declarations

Each architecture component has an associated service declaration. A service declaration represents how the resource communicates with other architecture resources and communication services. Each service declaration is specified with a Name, Type, and Value.

The service declarations for the components used in this diagram are already specified, so you don't need to specify them.

1. To view the service declarations, right click the block and choose *Properties* from the pop-up menu.
2. Click the *Service Declaration* tab.

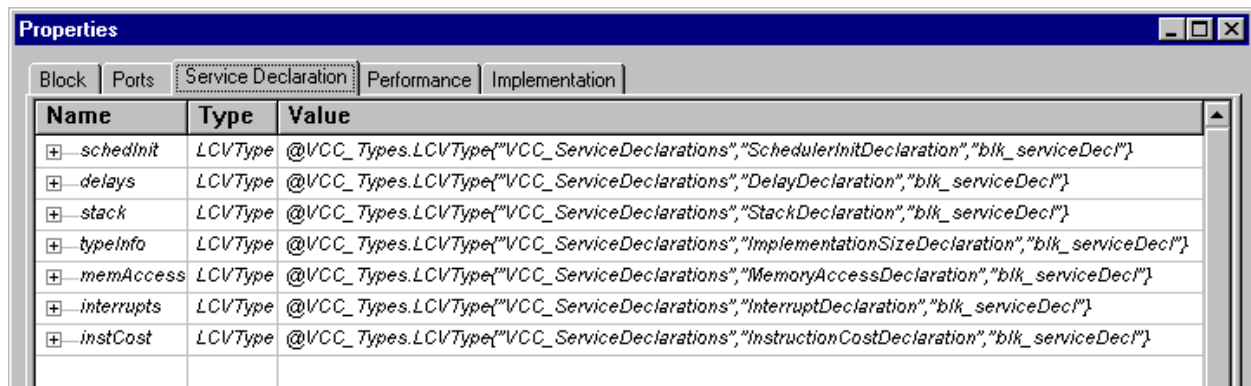
■ Service declarations for *MultipleServiceRTOS*



The screenshot shows the 'Properties' dialog box with the 'Service Declaration' tab selected. The table lists the following service declarations:

Name	Type	Value
+ libC	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","LibCDeclaration","blk_serviceDecl"}
+ schedInit	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SchedulerInitDeclaration","blk_serviceDecl"}
+ mutexes	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SWMutexDeclaration","blk_serviceDecl"}
+ interrupts	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SWInterruptDeclaration","blk_serviceDecl"}
+ timers	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SWTimerDeclaration","blk_serviceDecl"}

■ Service declarations for *SimpleCPU*



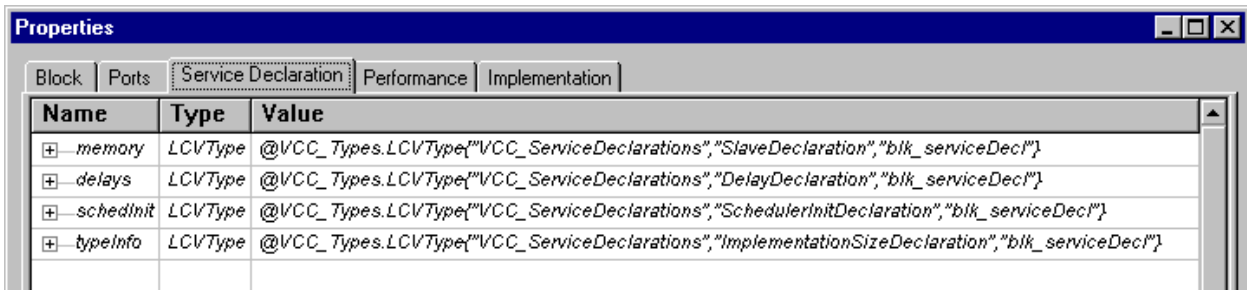
The screenshot shows the 'Properties' dialog box with the 'Service Declaration' tab selected. The table lists the following service declarations:

Name	Type	Value
+ schedInit	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SchedulerInitDeclaration","blk_serviceDecl"}
+ delays	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","DelayDeclaration","blk_serviceDecl"}
+ stack	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","StackDeclaration","blk_serviceDecl"}
+ typeInfo	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","ImplementationSizeDeclaration","blk_serviceDecl"}
+ memAccess	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","MemoryAccessDeclaration","blk_serviceDecl"}
+ interrupts	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","InterruptDeclaration","blk_serviceDecl"}
+ instCost	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","InstructionCostDeclaration","blk_serviceDecl"}

VCC Architecture Services Tutorial

Architecture Services Tutorial

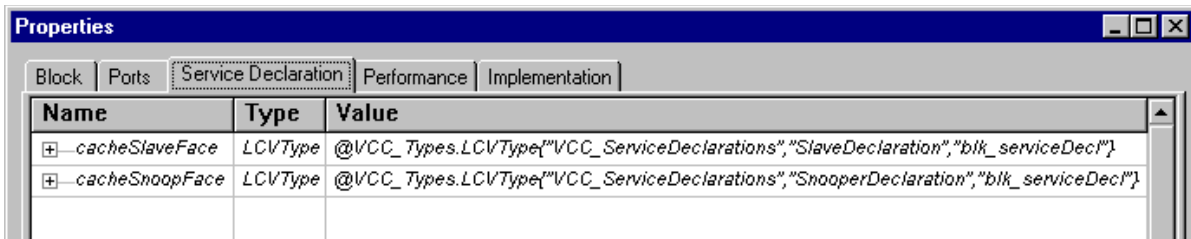
■ Service declarations for *SimpleASIC*



The screenshot shows the 'Properties' window with the 'Service Declaration' tab selected. It contains a table with four rows of service declarations for SimpleASIC.

Name	Type	Value
+ memory	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SlaveDeclaration","blk_serviceDecl"}
+ delays	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","DelayDeclaration","blk_serviceDecl"}
+ schedInit	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SchedulerInitDeclaration","blk_serviceDecl"}
+ typeInfo	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","ImplementationSizeDeclaration","blk_serviceDecl"}

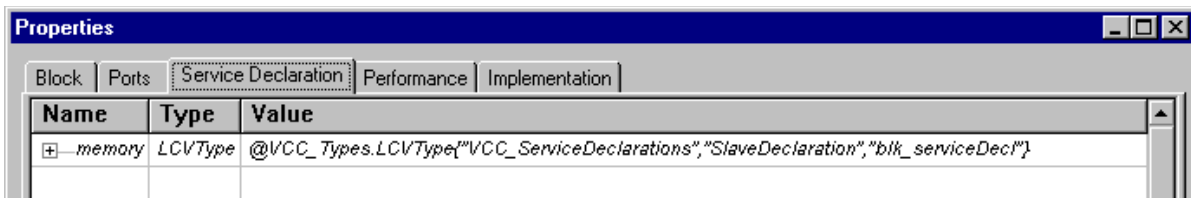
■ Service declarations for *SnoopyCache*



The screenshot shows the 'Properties' window with the 'Service Declaration' tab selected. It contains a table with two rows of service declarations for SnoopyCache.

Name	Type	Value
+ cacheSlaveFace	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SlaveDeclaration","blk_serviceDecl"}
+ cacheSnoopFace	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SnooperDeclaration","blk_serviceDecl"}

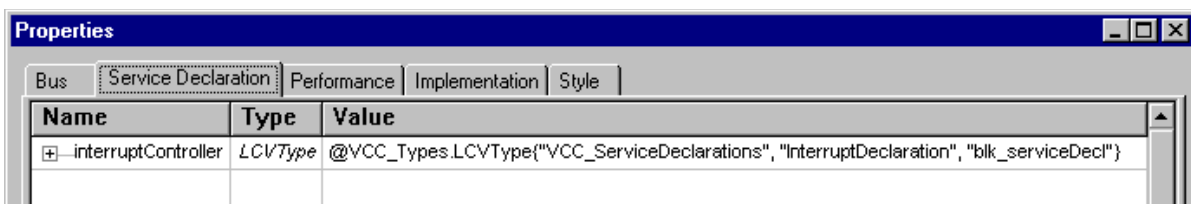
■ Service declarations for *SimpleMemory*



The screenshot shows the 'Properties' window with the 'Service Declaration' tab selected. It contains a table with one row of service declarations for SimpleMemory.

Name	Type	Value
+ memory	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","SlaveDeclaration","blk_serviceDecl"}

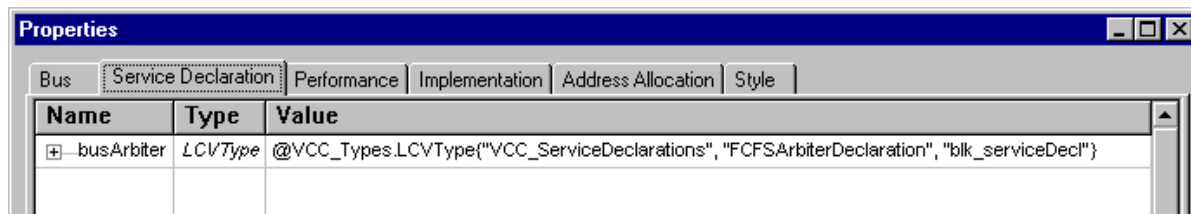
■ Service declarations for *InterruptBus*



The screenshot shows the 'Properties' window with the 'Service Declaration' tab selected. It contains a table with one row of service declarations for InterruptBus.

Name	Type	Value
+ interruptController	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","InterruptDeclaration","blk_serviceDecl"}

■ Service declarations for *FCFSBus* (both buses)



The screenshot shows the 'Properties' window with the 'Service Declaration' tab selected. It contains a table with one row of service declarations for FCFSBus.

Name	Type	Value
+ busArbiter	LCVType	@VCC_Types.LCVType{"VCC_ServiceDeclarations","FCFSArbiterDeclaration","blk_serviceDecl"}

Architecture Services

Each architecture component view has an associated service binding. Each service binding is specified with a Name, Type, and Value. The Name is the same name you saw in the “Service Declarations” section.

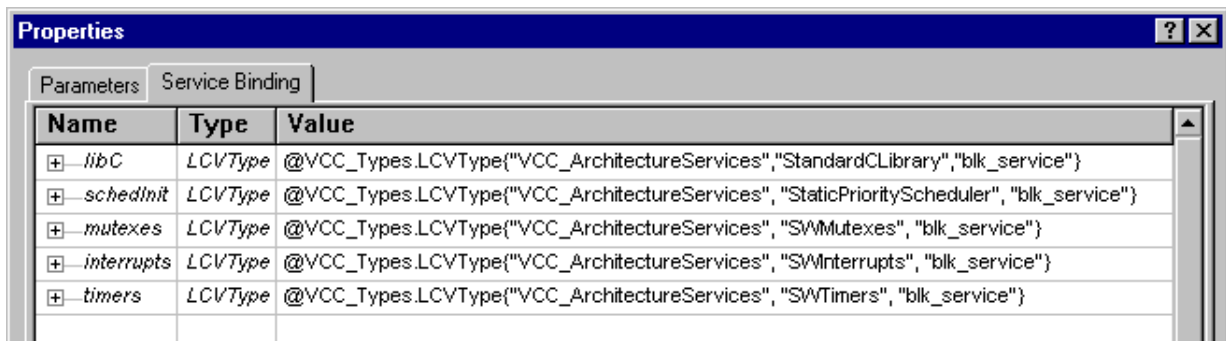
The service bindings for the components used in this diagram are already specified, so you don’t need to edit them.

1. To view the service bindings, right click the block and choose *Edit View* from the pop-up menu.
2. In the Edit View dialog, select the view whose service bindings you want to view and click *OK*.

The service bindings shown here are for the views you specified in the “Bind the Views” section.

3. Click the *Service Binding* tab.

■ Architecture service bindings for *MultipleServiceRTOS*



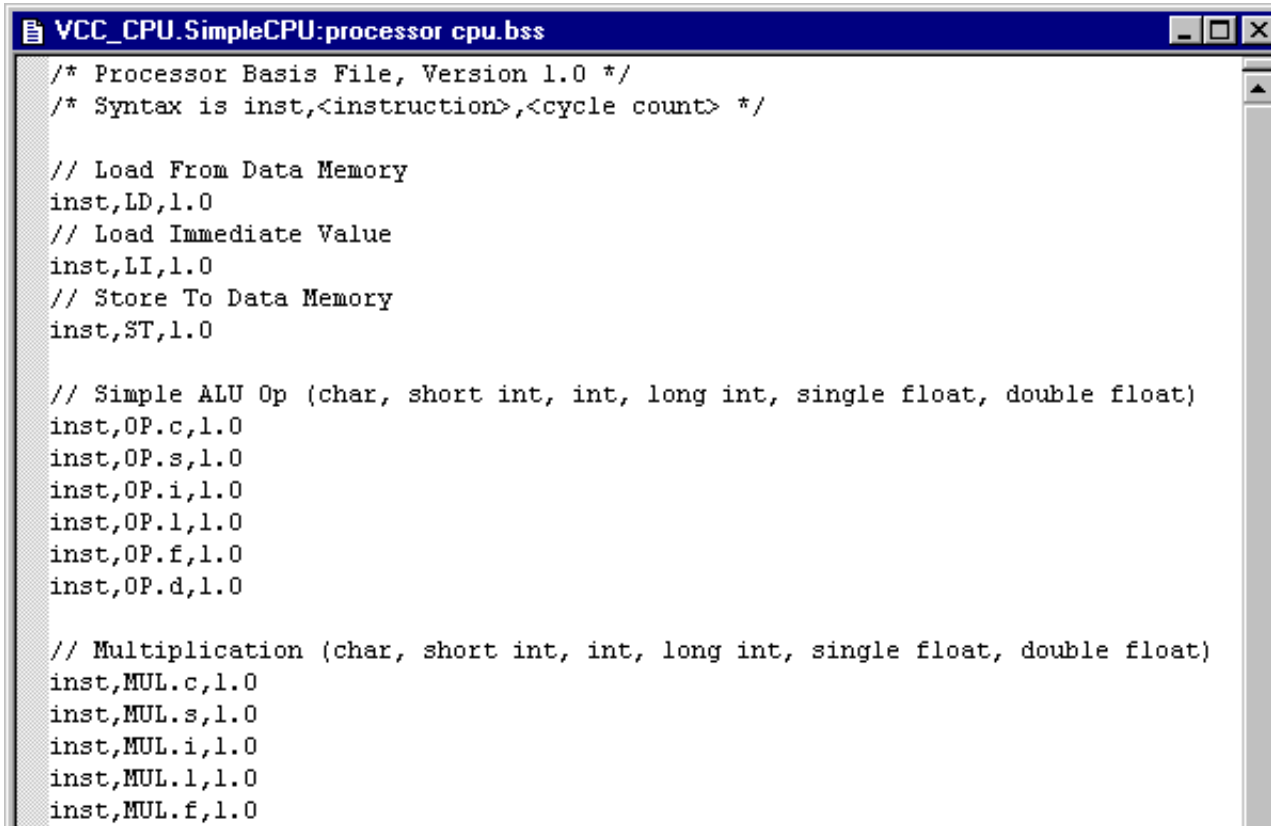
The screenshot shows a 'Properties' dialog box with the 'Service Binding' tab selected. It contains a table with three columns: Name, Type, and Value. The table lists five service bindings for the 'MultipleServiceRTOS' component.

Name	Type	Value
<input type="checkbox"/> libC	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "StandardCLibrary", "blk_service")
<input type="checkbox"/> schedInit	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "StaticPriorityScheduler", "blk_service")
<input type="checkbox"/> mutexes	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "SVMutexes", "blk_service")
<input type="checkbox"/> interrupts	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "SVMInterrupts", "blk_service")
<input type="checkbox"/> timers	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "SVMTimers", "blk_service")

VCC Architecture Services Tutorial

Architecture Services Tutorial

■ Architecture service bindings for *SimpleCPU*



The screenshot shows a VCC editor window titled "VCC_CPU.SimpleCPU:processor cpu.bss". The code inside is a processor basis file for SimpleCPU, version 1.0. It defines instructions for loading from data memory, loading immediate values, storing to data memory, and performing various ALU operations (comparison, shift, integer, long integer, single float, double float) and multiplication (comparison, shift, integer, long integer, single float, double float).

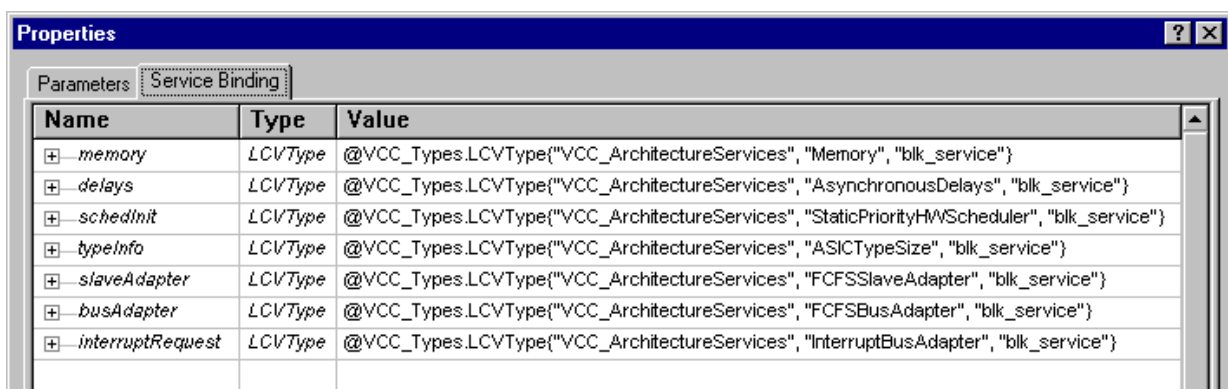
```
/* Processor Basis File, Version 1.0 */
/* Syntax is inst,<instruction>,<cycle count> */

// Load From Data Memory
inst,LD,1.0
// Load Immediate Value
inst,LI,1.0
// Store To Data Memory
inst,ST,1.0

// Simple ALU Op (char, short int, int, long int, single float, double float)
inst,OP.c,1.0
inst,OP.s,1.0
inst,OP.i,1.0
inst,OP.l,1.0
inst,OP.f,1.0
inst,OP.d,1.0

// Multiplication (char, short int, int, long int, single float, double float)
inst,MUL.c,1.0
inst,MUL.s,1.0
inst,MUL.i,1.0
inst,MUL.l,1.0
inst,MUL.f,1.0
```

■ Architecture service bindings for *SimpleASIC*



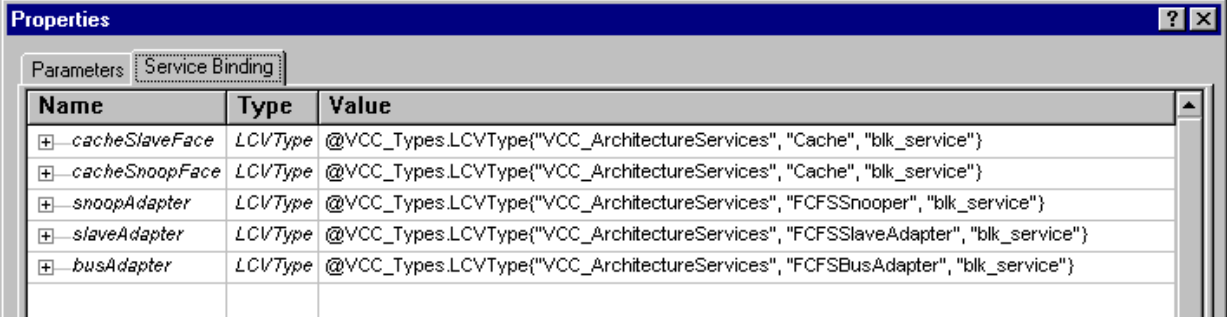
The screenshot shows the Properties dialog box for SimpleASIC, with the Service Binding tab selected. The table lists the service bindings for various components.

Name	Type	Value
memory	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "Memory", "blk_service")
delays	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "AsynchronousDelays", "blk_service")
schedinit	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "StaticPriorityHWScheduler", "blk_service")
typeinfo	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "ASICTypeSize", "blk_service")
slaveAdapter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSSlaveAdapter", "blk_service")
busAdapter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSBusAdapter", "blk_service")
interruptRequest	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "InterruptBusAdapter", "blk_service")

VCC Architecture Services Tutorial

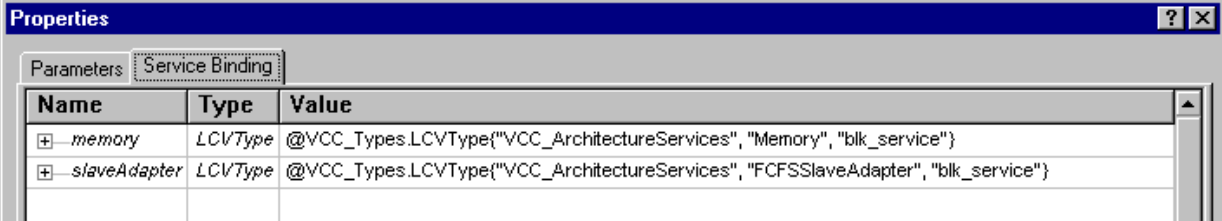
Architecture Services Tutorial

■ Architecture service bindings for *SnoopyCache*



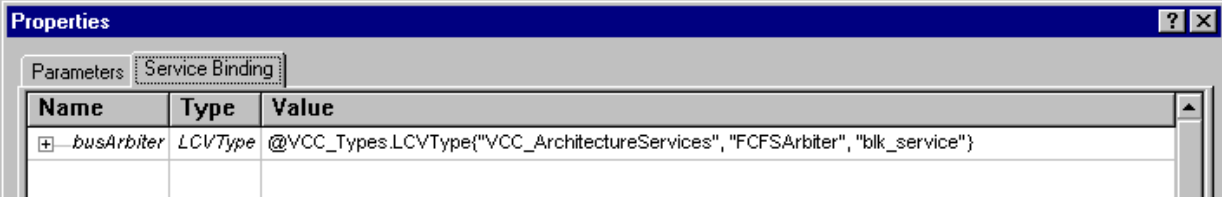
Parameters Service Binding		
Name	Type	Value
+ cacheSlaveFace	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "Cache", "blk_service")
+ cacheSnoopFace	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "Cache", "blk_service")
+ snoopAdapter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSSnooper", "blk_service")
+ slaveAdapter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSSlaveAdapter", "blk_service")
+ busAdapter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSBusAdapter", "blk_service")

■ Architecture service bindings for *SimpleMemory*



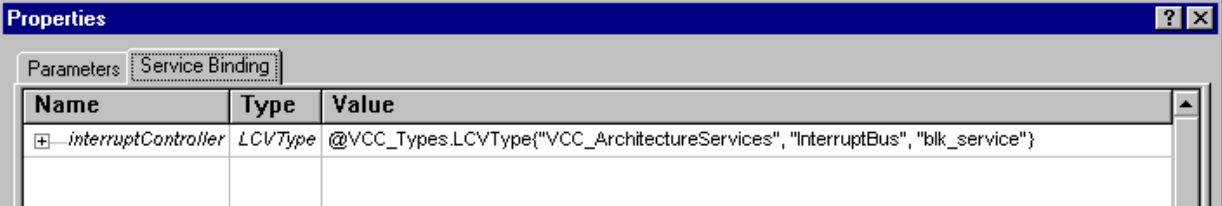
Parameters Service Binding		
Name	Type	Value
+ memory	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "Memory", "blk_service")
+ slaveAdapter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSSlaveAdapter", "blk_service")

■ Architecture service bindings for *FCFSBus* (both buses)



Parameters Service Binding		
Name	Type	Value
+ busArbiter	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "FCFSArbiter", "blk_service")

■ Architecture service bindings for *InterruptBus*



Parameters Service Binding		
Name	Type	Value
+ interruptController	LCVType	@VCC_Types.LCVType("VCC_ArchitectureServices", "InterruptBus", "blk_service")

Specify Parameter Values

There are many parameters associated with an architecture block. In this section, you will set the parameter values for your blocks to the values shown in the following examples.

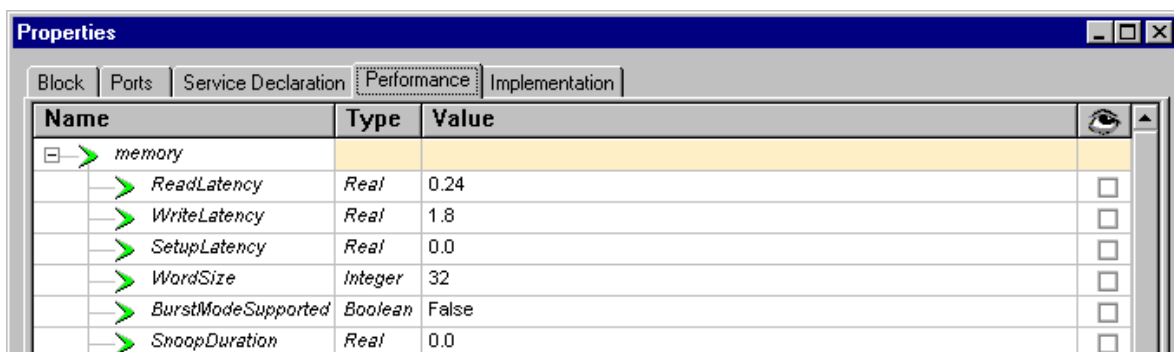
1. To display the parameters, right click the block and choose *Properties* from the pop-up menu.
2. Click the *Performance* tab in the Properties dialog.
3. To expand the parameters, click the plus sign before the parameter group headers.
4. To edit parameter values, double-click in the Value field.
 - ❑ If the value is a number, type the new number.
 - ❑ If the value is a library.cell:view, use the browse button (...) to locate the lib.cell:view.
 - ❑ If the value is Boolean, you can type *True* or *False* or choose from the down arrow.
 - ❑ Some parameter values look like this:

```
@VCC_Types.SchedulerOverheadType{0.0,0.0,0.0,0.0}
```

These values represent the heading for a group of values. When you enter the individual values under that heading, the values are appended together and displayed with the group heading. You do not need to enter the values for the group heading.

5. To save the parameter values and close the Properties dialog, click *OK*.
6. Set the parameter values for each block as shown in the following examples.

■ Parameters for *SimpleMemory*



Properties				
Block Ports Service Declaration Performance Implementation				
Name	Type	Value		
memory				
ReadLatency	Real	0.24		
WriteLatency	Real	1.8		
SetupLatency	Real	0.0		
WordSize	Integer	32		
BurstModeSupported	Boolean	False		
SnoopDuration	Real	0.0		

VCC Architecture Services Tutorial

Architecture Services Tutorial

■ Parameters for *MultipleServiceRTOS*

Properties			
Block Ports Service Declaration Performance Implementation			
Name	Type	Value	
[-] > fifo_sched			
> TickISRInterruptNumber	Unsigned	1	<input type="checkbox"/>
> MaximumNumberOfTimers	Unsigned	4	<input type="checkbox"/>
> TickISROverhead	Real	0.0	<input type="checkbox"/>
> TickHandlerOverhead	Real	0.0	<input type="checkbox"/>
> TimerResolution	Integer	0	<input type="checkbox"/>
> UseHWTimer	Boolean	False	<input type="checkbox"/>
[-] > SchedulerOverhead	SchedulerOverheadType	@VCC_Types.SchedulerOverheadType{0.0,0.0,0.0,0.0}	<input type="checkbox"/>
> start_overhead	real	0.0	
> finish_overhead	real	0.0	
> suspend_overhead	real	0.0	
> resume_overhead	real	0.0	
> MaximumNumberOfMutexes	Unsigned	4	<input type="checkbox"/>
> LargerPriorityIsHigherPriority	Boolean	True	<input type="checkbox"/>
> SchedulerPreemptionStyle	SchedulerPreemptionStyleType	Suspend	<input type="checkbox"/>
[-] > rr_sched			
> TickISRInterruptNumber	Unsigned	2	<input type="checkbox"/>
> MaximumNumberOfTimers	Unsigned	4	<input type="checkbox"/>
> TickISROverhead	Real	0.0	<input type="checkbox"/>
> TickHandlerOverhead	Real	0.0	<input type="checkbox"/>
> TimerResolution	Integer	0	<input type="checkbox"/>
> UseHWTimer	Boolean	False	<input type="checkbox"/>
[-] > SchedulerOverhead	SchedulerOverheadType	@VCC_Types.SchedulerOverheadType{0.0,0.0,0.0,0.0}	<input type="checkbox"/>
> start_overhead	real	0.0	
> finish_overhead	real	0.0	
> suspend_overhead	real	0.0	
> resume_overhead	real	0.0	
> MaximumNumberOfMutexes	Unsigned	4	<input type="checkbox"/>
> Quantum	Real	0.0	<input type="checkbox"/>

Click the plus signs to expand these headings.
Do not enter values for these parameter headings.

Double-click the value field, click the down arrow, then select *Suspend*.

VCC Architecture Services Tutorial

Architecture Services Tutorial

■ Parameters for *SimpleCPU*

Properties			
Block Ports Service Declaration Performance Implementation			
Name	Type	Value	
processor			
ICOn	Boolean	True	
ICPrioritiesOn	Boolean	False	
MergeTransactions	Boolean	False	
MergeTransactionSegmentBufferSize	Integer	1	
TypeSizeSpecification	@VCC_Types.SizeOfSpecification	{4,8,1,4,2,4,4,""}	
int	Unsigned	4	
double	Unsigned	8	
char	Unsigned	1	
long	Unsigned	4	
short	Unsigned	2	
pointer	Unsigned	4	
alignment	Unsigned	4	
view	String	"	
BusAdapterPriority	Unsigned	0	
NumberOfIRQs	Unsigned	4	
MaxTimeSpanForMergedTransactions	Real	0.0	
MaxBytesOfMergedTransactions	Integer	0	
CPS	Real	1000000.0	
BitsPerByte	Unsigned	8	
WordSize	Unsigned	4	
CommunicationMode	@VCC_Types.MemoryAccessPerfMode	BusTraffic	
DataWriteMode	@VCC_Types.MemoryAccessPerfMode	BusTraffic	
DataReadMode	@VCC_Types.MemoryAccessPerfMode	BusTraffic	
InstructionMode	@VCC_Types.MemoryAccessPerfMode	BusTraffic	
StackSize	Unsigned	1024	

Click the plus sign to expand this heading. Do not enter values for this parameter heading.

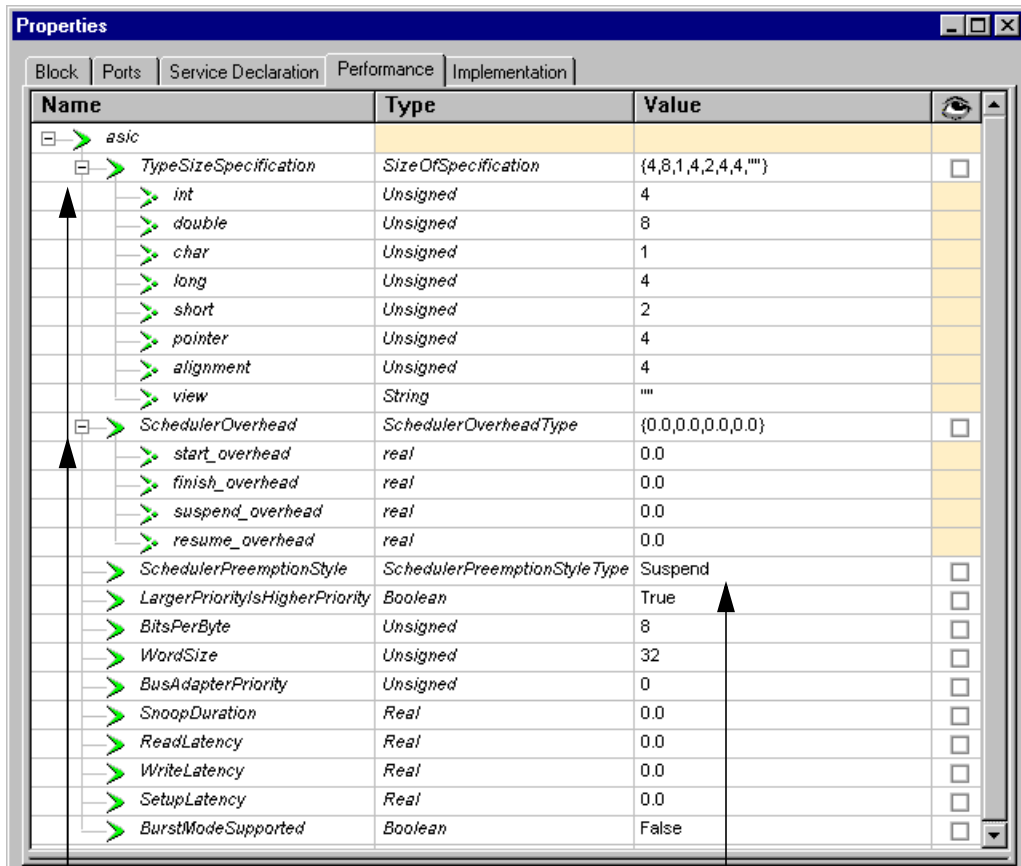
Double-click the value field, click the down arrow, then select *BusTraffic*.

This is an empty string. Type an opening and closing quotation mark.

VCC Architecture Services Tutorial

Architecture Services Tutorial

■ Parameters for *SimpleASIC*



Name	Type	Value
asic		
TypeSizeSpecification	SizeOfSpecification	{4,8,1,4,2,4,4,""}
int	Unsigned	4
double	Unsigned	8
char	Unsigned	1
long	Unsigned	4
short	Unsigned	2
pointer	Unsigned	4
alignment	Unsigned	4
view	String	""
SchedulerOverhead	SchedulerOverheadType	{0.0,0.0,0.0,0.0}
start_overhead	real	0.0
finish_overhead	real	0.0
suspend_overhead	real	0.0
resume_overhead	real	0.0
SchedulerPreemptionStyle	SchedulerPreemptionStyleType	Suspend
LargerPriorityIsHigherPriority	Boolean	True
BitsPerByte	Unsigned	8
WordSize	Unsigned	32
BusAdapterPriority	Unsigned	0
SnoopDuration	Real	0.0
ReadLatency	Real	0.0
WriteLatency	Real	0.0
SetupLatency	Real	0.0
BurstModeSupported	Boolean	False

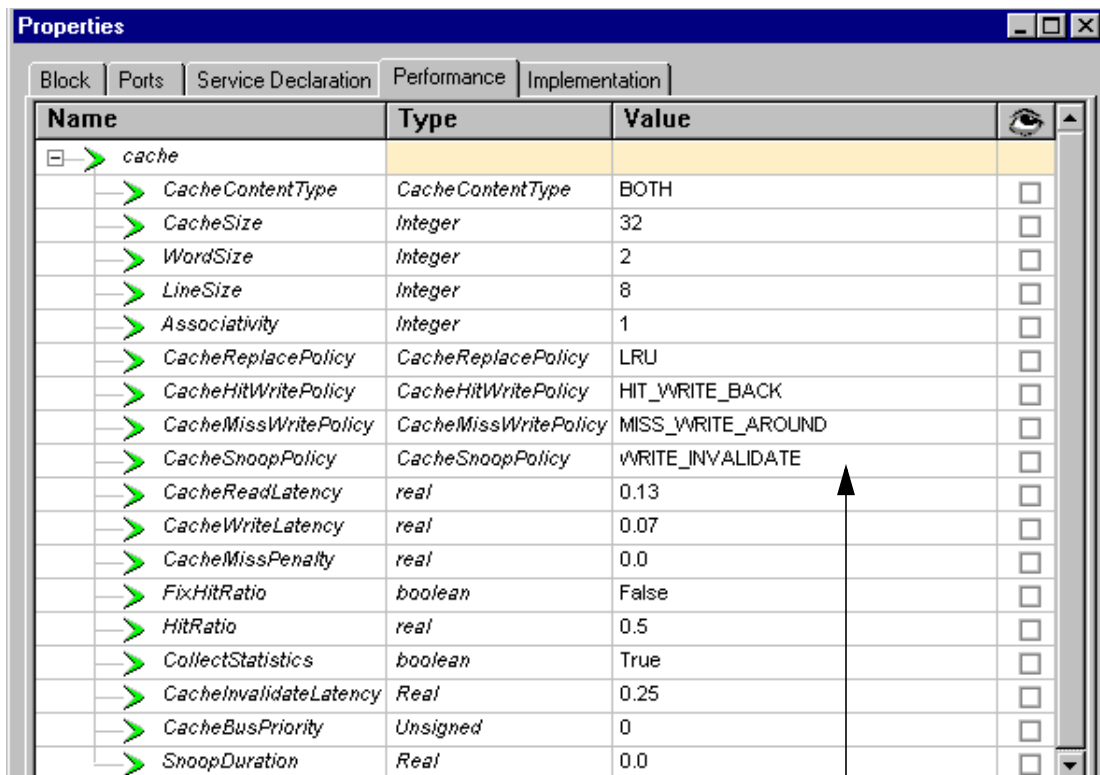
Double-click the value field, click the down arrow, then select *Suspend*.

Click the plus signs to expand these headings. Do not enter values for these parameter headings.

VCC Architecture Services Tutorial

Architecture Services Tutorial

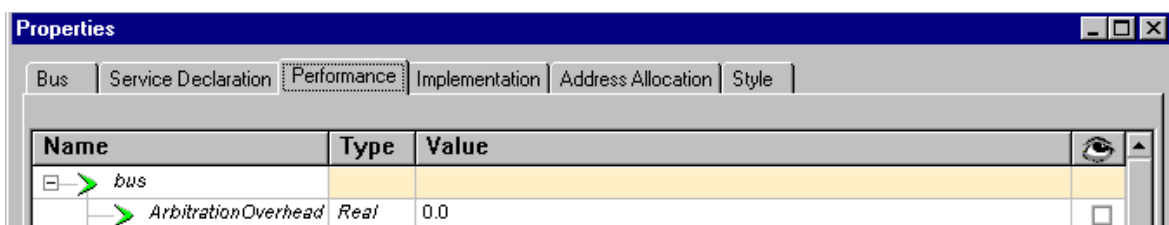
■ Parameters for *SnoopyCache*



Name	Type	Value	
cache			
CacheContentType	CacheContentType	BOTH	
CacheSize	Integer	32	
WordSize	Integer	2	
LineSize	Integer	8	
Associativity	Integer	1	
CacheReplacePolicy	CacheReplacePolicy	LRU	
CacheHitWritePolicy	CacheHitWritePolicy	HIT_WRITE_BACK	
CacheMissWritePolicy	CacheMissWritePolicy	MISS_WRITE_AROUND	
CacheSnoopPolicy	CacheSnoopPolicy	WRITE_INVALIDATE	
CacheReadLatency	real	0.13	
CacheWriteLatency	real	0.07	
CacheMissPenalty	real	0.0	
FixHitRatio	boolean	False	
HitRatio	real	0.5	
CollectStatistics	boolean	True	
CacheInvalidateLatency	Real	0.25	
CacheBusPriority	Unsigned	0	
SnoopDuration	Real	0.0	

Double-click the value field, click the down arrow, then select the value.

■ Parameters for *FCFSBus* (both FCFS buses)



Name	Type	Value	
bus			
ArbitrationOverhead	Real	0.0	

■ There are no parameters for the *InterruptBus*.

7. To save the *Tutorial.arch_system:arch* diagram, choose *File > Save*.

Note: If you were unable to specify the parameters for these architecture blocks:

- ❑ Close your *Tutorial.arch_system:arch* cellview
- ❑ Open *Tutorial_Examples.Example6:arch*
- ❑ Save the example as *Tutorial.arch_system:arch*

Generate a Netlist

The *Generate Netlist* function verifies the syntax of your code and reports any errors.

1. To generate and verify the code for this architecture diagram, choose *Tools > Generate Netlist*.

The text field at the bottom of the VCC window reports that the code generation was successful or it reports the cause of any errors.

2. Ignore the following messages that the Child port on the RTOS is unconnected.

```
Warning: The port 'Child' is not connected.  
{in /(Tutorial.arch_system:arch)I_0.Child}:
```

```
Warning: Some ports in the instance 'MultipleServiceRTOS(I_0)'  
are not connected. {in /(Tutorial.arch_system:arch)I_0}.
```

☐ *Warnings* do not cause the code generation to fail.

☐ *Errors* cause the code generation to fail.

- **Successful code generation message:**

```
Info: Code generation for Tutorial.arch_system:arch was successful.
```

- **Unsuccessful code generation message (caused by a missing parameter value):**

```
Error: The parameter "<parameter name>" has no value.  
{in /(Tutorial.arch_system:arch)I_0}.
```

This error message is only a sample. You might see other types of messages, but the information in the message should help you correct your design.

Note: If you don't know which block is causing the problem (in this example error message, it's I_0), right click on a block and choose *Properties*. The *Block* tab contains information about the Instance Name.

3. Correct any errors and run *Tools > Generate Netlist* until the code generation is successful.

Note: If you are unable to generate the code for the architecture model:

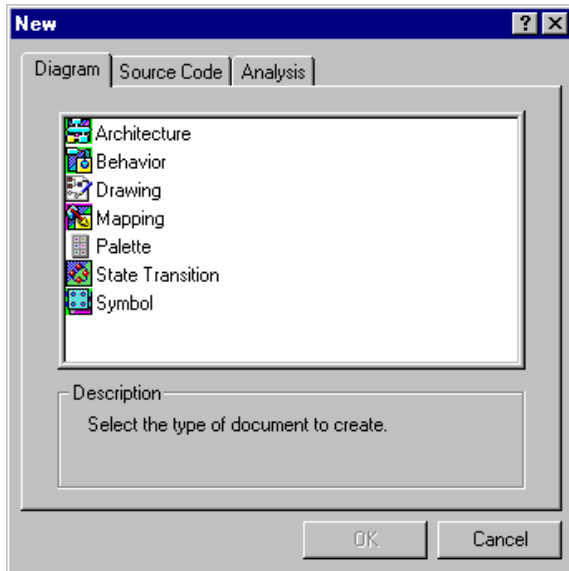
- ☐ Close your *Tutorial.arch_system:arch* cellview
- ☐ Open *Tutorial_Examples.Example6:arch*
- ☐ Save the example as *Tutorial.arch_system:arch*

4. To save the architecture model, choose *File > Save*.
5. To close the architecture model, choose *File > Close*.

Create a Mapping Diagram

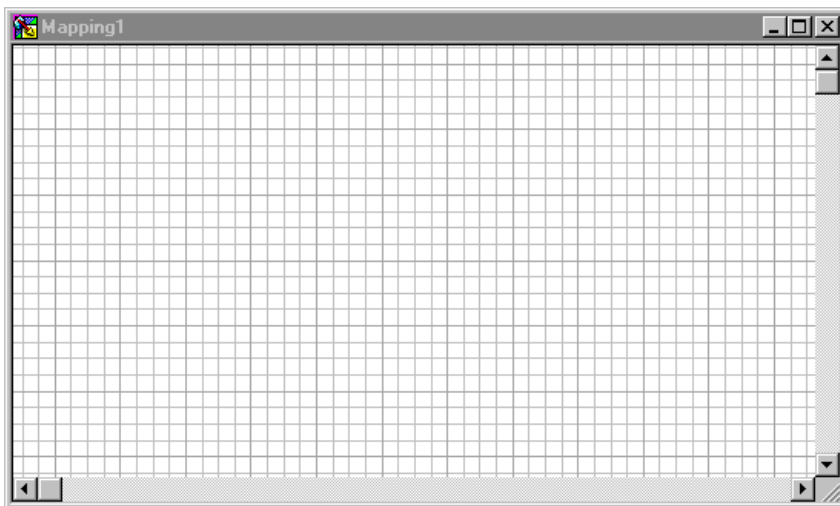
A mapping diagram lets you map your behavior diagram to your architecture diagram, simulate the performance, and analyze the results.

1. Choose *File > New*.



2. Choose *Mapping* and click *OK*.

A new empty design titled *Mapping1* opens.

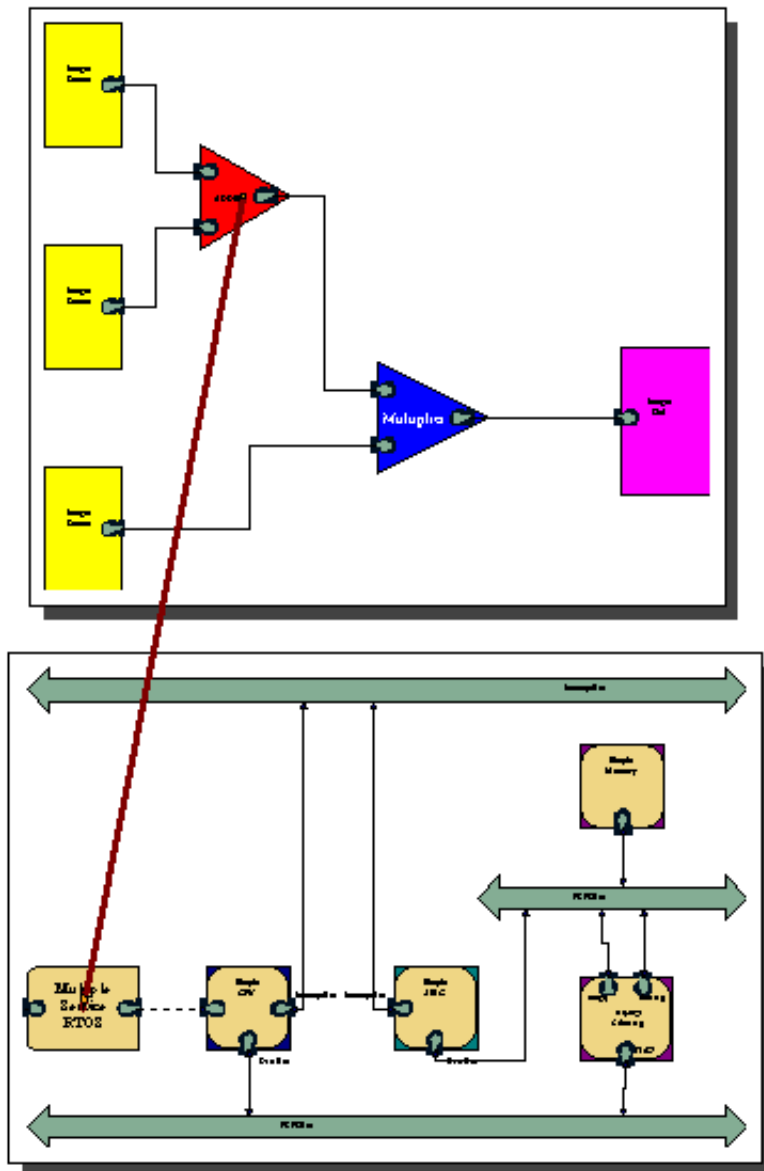


Note: To toggle the grid on and off, choose *View > Grid*.

Add Mapping Connections

1. Choose *Mapping > Mapping Connection*.
2. To draw a mapping connection from the *Adder* to the *RTOS*, click on the *Adder* in the behavior diagram, then click on the *RTOS* in the architecture diagram.

Your diagram now looks like this:

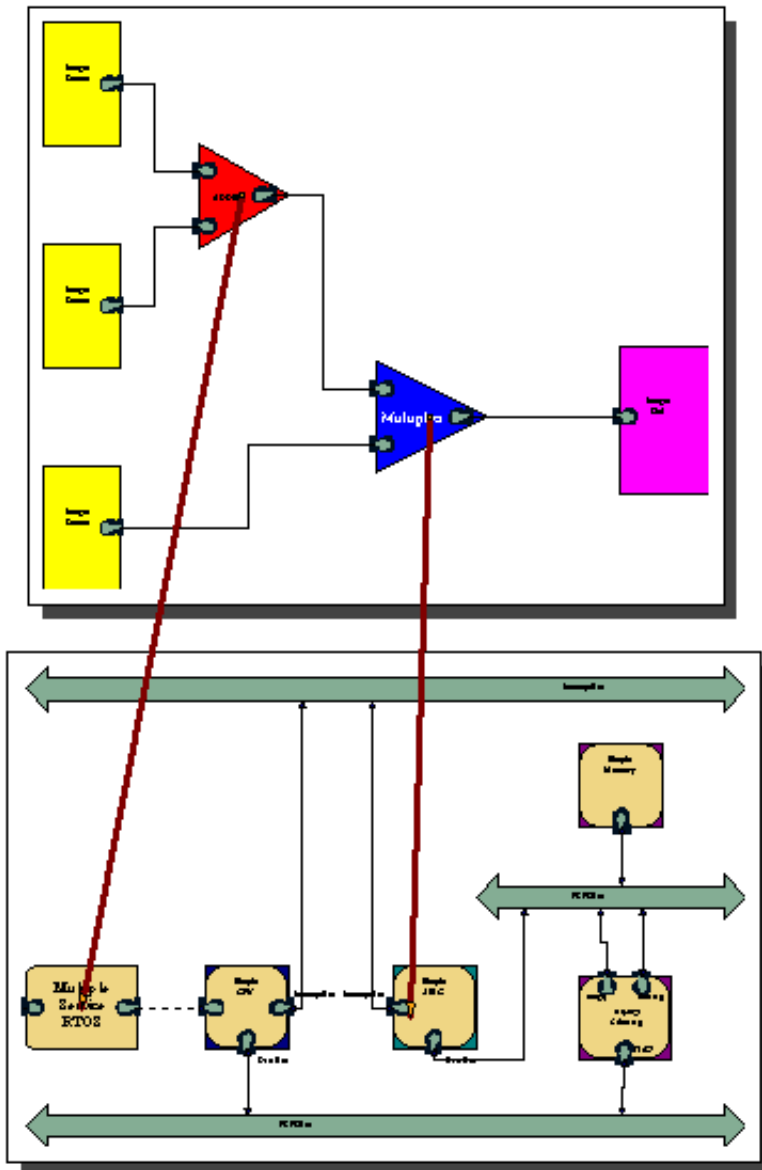


VCC Architecture Services Tutorial

Architecture Services Tutorial

3. Choose *Mapping > Mapping Connection*.
4. To draw a mapping connection from the *Multiplier* to the *ASIC*, click on the *Multiplier* in the behavior diagram, then click on the *ASIC* in the architecture diagram.

Your diagram now looks like this:



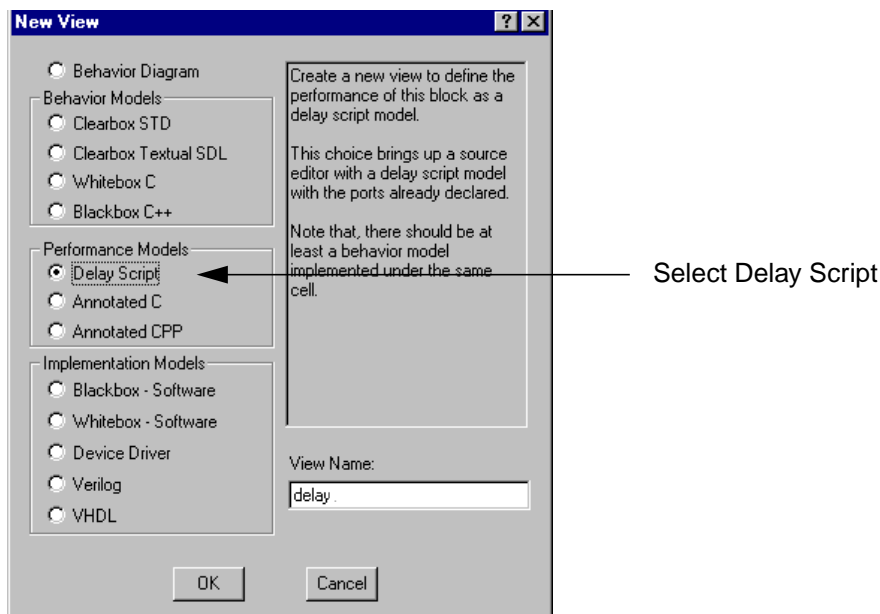
Add Performance Views

Important

Under no circumstances can you make a copy of *Example7* and skip steps 1-6. You can avoid all other steps in the “Creating a Mapping Diagram” section by using the examples in the *Tutorial_Examples* library, but you **MUST** perform steps 1-6 yourself.

You will now add a new DSL (delay scripting language) performance view to the *Adder* behavior block and a delay performance view to the connection between the *Adder* and the *RTOS*.

1. To add a new performance view to the *Adder*, right click on the *Adder* symbol, and choose *New View* from the pop-up menu.



2. In the Performance Models section, choose *Delay Script*, then click *OK*.

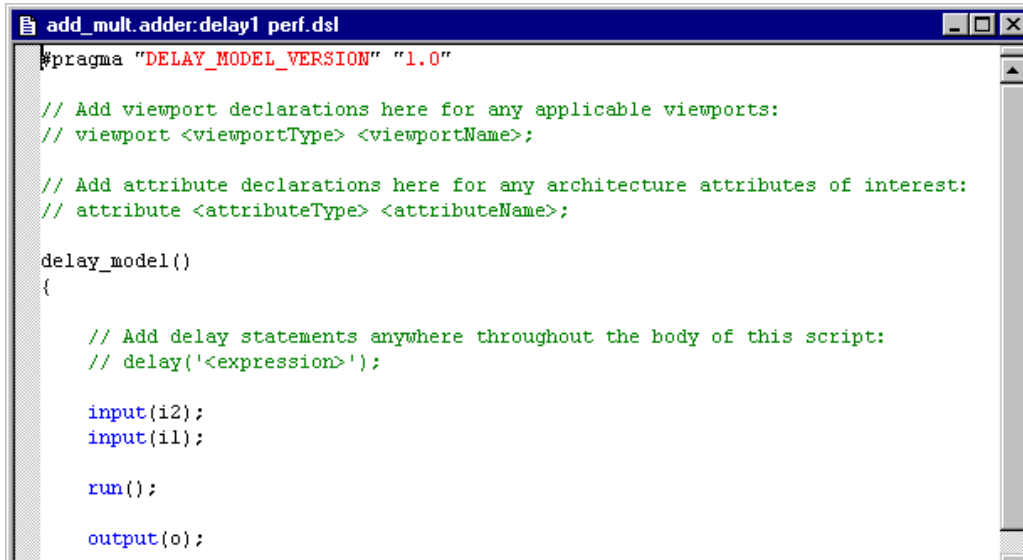
A dialog ask if you want to update the interface of the adder.

3. Click Yes in the dialog.

VCC Architecture Services Tutorial

Architecture Services Tutorial

The delay scripting language file (*perf.dsl*) for the performance model opens:



```
#pragma "DELAY_MODEL_VERSION" "1.0"

// Add viewport declarations here for any applicable viewports:
// viewport <viewportType> <viewportName>;

// Add attribute declarations here for any architecture attributes of interest:
// attribute <attributeType> <attributeName>;

delay_model()
{
    // Add delay statements anywhere throughout the body of this script:
    // delay('<expression>');

    input(i2);
    input(i1);

    run();

    output(o);
}
```

4. Add the delay expression *delay('10.0')*, so the script looks like this:

```
delay_model()
{
    // Add delay statements anywhere throughout
    // delay('<expression>');

    input(i2);
    input(i1);

    run();
    delay('10.0');
    output(o);
}
```

← Add this delay expression

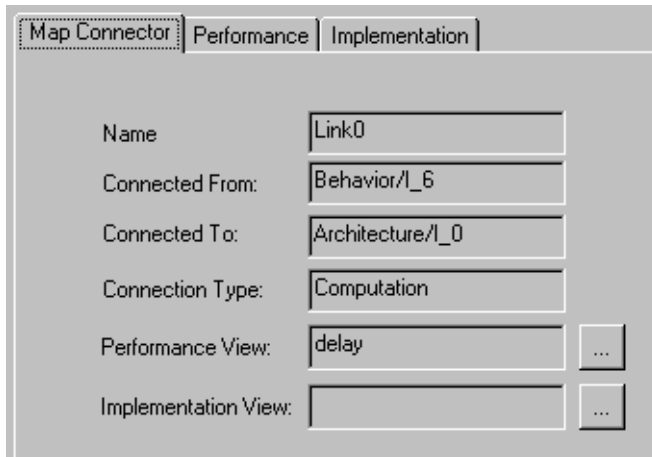
5. To save the *perf.dsl* file, make sure the *perf.dsl* file is the active file, and choose *File > Save*.
6. To close the *perf.dsl* file, choose *File > Close*.
7. To set the performance view for the mapping connections, select the connection wire between the *Adder* and the *RTOS*.

Make sure you see the green markers at each end of the wire.

VCC Architecture Services Tutorial

Architecture Services Tutorial

8. Right click and choose *Properties* from the pop-up menu.



9. Click the browse button (...) next to the Performance View field.
10. Click the down arrow for Performance View and choose *delay*.
11. Click *OK* in the Bind View dialog, then click *OK* in the Properties dialog.

Specify a Memory Segment Parameter

The *SimpleCPU* requires a Memory Segment parameter. The value for this parameter is the instance name of the *SimpleMemory* block.

1. To find the instance name of the *SimpleMemory* block, right click on the block and choose *Properties*.

Note: Look for the green box around the *SimpleMemory* block to make sure you've selected the block and not a port or the entire model.

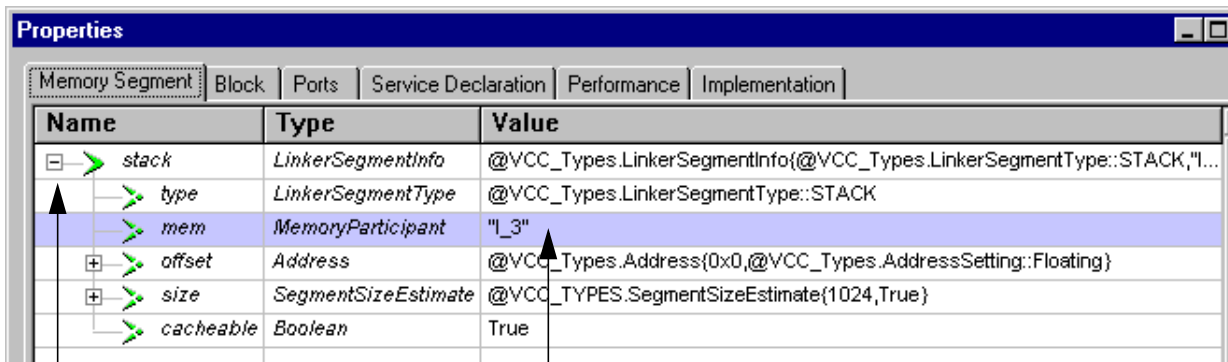
The *Block* tab contains information about the Instance Name. For this example, the instance name is `I_3`.

2. Make a note of the instance name of the *SimpleMemory* block.

You will use this instance name for the *mem* parameter in step 6.

3. To close the Properties form, click *Cancel*.
4. To specify a Memory Segment parameter for the *SimpleCPU*, right click on the *SimpleCPU* architecture block, and select *Properties* from the pop-up menu.

Note: Look for the green box around the *SimpleCPU* block to make sure you've selected the block and not a port or the entire model.



Click the plus sign to expand this parameter heading.

Enter the value "I_3", including the opening and closing quotation mark.

5. Click the *Memory Segment* tab, and expand the *stack* parameter heading.
6. Enter the instance name of the *SimpleMemory* block as the value for the *mem* parameter.

You must include the opening and closing quotation marks in the value. For example `"I_3"`.

VCC Architecture Services Tutorial

Architecture Services Tutorial

7. To save the parameter values and close the Properties dialog, click *OK*.
8. To save the Mapping diagram, choose *File > Save*.
9. Double-click the *Tutorial* library icon in the browser.
10. Specify the file name as *mapping_config*, and click *Save*.
11. When you are asked if you want to save the configuration for this design, click *Yes*.

Save the configuration as *Tutorial.mapping_config:config*.

Note: If you are unable to complete the mapping diagram:

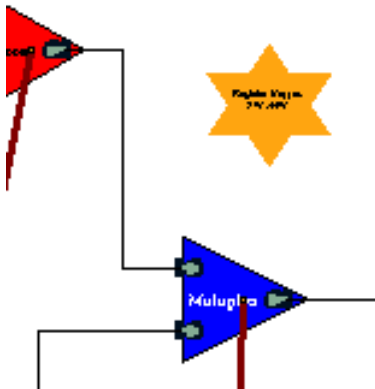
- ☐ Close your *Tutorial.mapping_config:mapping* cellview
- ☐ Open *Tutorial_Examples.Example7:mapping*
- ☐ Save the example as *Tutorial.mapping_config:mapping*

Add a Mapping Pattern

To refine your simulation, you can map a communication wire to a communication pattern that models specific protocols between the sender and receiver. To transfer a small data token, you can use the *RegisterMappedSwHw* pattern in which the data is transferred directly to a register on the ASIC.

1. To instantiate the *RegisterMappedSwHw* symbol in this mapping diagram, do the following:
 - ❑ In the Project Folder, expand the *VCC_Patterns* library, then expand the *RegisterMappedSwHw* cell.
 - ❑ Click the *RegisterMappedSwHw* symbol view and drag it to the behavior model in the mapping diagram.
2. Place this symbol near the upper input port of the *Multiplier*.

Your diagram now looks like this:

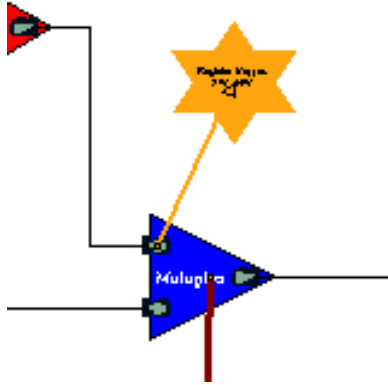


3. Choose *Mapping > Mapping Connection*.
4. To draw the mapping connection, click on the upper input port of the *Multiplier*, then click on the *RegisterMappedSwHw*.

VCC Architecture Services Tutorial

Architecture Services Tutorial

Your diagram now looks like this:



5. Right click on the *RegisterMappedSwHw* symbol and choose *Bind View* from the pop-up menu.
6. Choose *proto* and click *OK*.

The *proto* view models the refined simulation of the protocol.

7. To save the Mapping diagram, choose *File > Save*.

Note: If you are unable to add the mapping pattern:

- ☐ Close your *Tutorial.mapping_config:mapping* cellview
- ☐ Open *Tutorial_Examples.Example8:mapping*
- ☐ Save the example as *Tutorial.mapping_config:mapping*

Generate a Netlist

1. To generate a netlist for the mapping diagram, choose *Tools > Generate Netlist*.

- Ignore the following messages that the Child port on the RTOS is unconnected.

```
Warning: The port 'Child' is not connected.  
{in /(Tutorial.arch_system:arch)I_0.Child}:
```

```
Warning: Some ports in the instance 'MultipleServiceRTOS(I_0)'  
are not connected. {in /(Tutorial.arch_system:arch)I_0}.
```

- ☐ *Warnings* do not cause the netlist generation to fail.
- ☐ *Errors* cause the netlist generation to fail.

- Successful netlist generation message:

```
Info: Code generation for Tutorial.behav_system:behav was successful.  
Info: Code generation for Tutorial.arch_system:arch was successful.  
Info: Code generation for Tutorial.mapping_config:mapping was successful.
```

- Unsuccessful netlist generation message (caused by a missing parameter value):

```
Error: The parameter "<parameter name>" has no value.  
{in /(Tutorial.arch_system:arch)I_0}.
```

This error message is only a sample. You might see other types of messages, but the information in the message should help you correct your design.

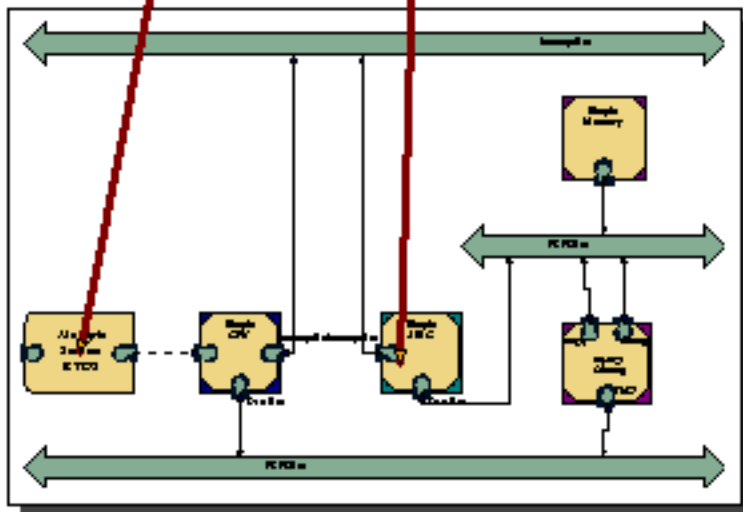
Note: If you don't know which block is causing the problem (in this example error message, it's I_0), right click on a block and choose *Properties*. The *Block* tab contains information about the Instance Name.

2. Correct any errors and run *Tools > Generate Netlist* until the code generation is successful.

Note: If you are unable to successfully generate your code:

- ☐ Close your *Tutorial.mapping_config:mapping* cellview
- ☐ Open *Tutorial_Examples.Example8:mapping*
- ☐ Save the example as *Tutorial.mapping_config:mapping*

3. To save the mapping diagram, choose *File > Save*.

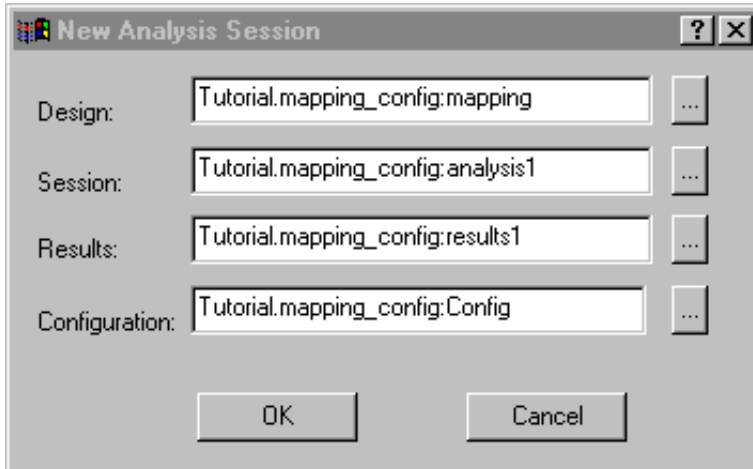


- Product Version 2.1

VCC Architecture Services Tutorial

Architecture Services Tutorial

The New Analysis Session dialog opens.



3. Accept the default names for the simulation session and results file name, and click *OK*.

Notice that the *Analysis* menu is added to the main menu bar.

4. Choose *Analysis > Initialize*.

A dialog prompts you to enter an end time for the simulation.

5. Set the value for End Time to 1000, then click *OK*.

1000 is a safe value because the memory-related transactions of the simulation will complete within this time.

An Analysis Cockpit dialog asks if you want to save the session changes.

6. Click *Yes* in the Analysis Cockpit dialog.

As VCC initializes the simulation, it reports the progress in the text window at the bottom of the screen.

7. When the text window reports *Info:Done*, choose *Analysis > Go/Continue*.

VCC reports the progress in the text window at the bottom of the screen.

8. To view the output of the analysis, open the following directory:

```
C:\MyWorkspace\Tutorial\mapping_config\results
```

This directory contains several error, debugging, and information files.

VCC Architecture Services Tutorial

Architecture Services Tutorial

Note: If you are unable to successfully generate your code:

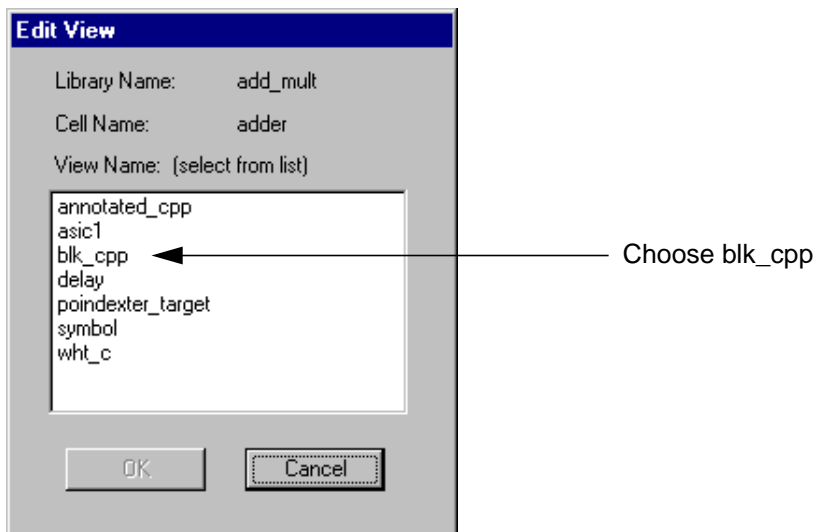
- ❑ Close your *Tutorial.mapping_config:mapping* cellview
- ❑ Open *Tutorial_Examples.Example8:mapping*
- ❑ Save the example as *Tutorial.mapping_config:mapping*
- ❑ Rerun the simulation.

Supplemental Exercise: Add Memory Reference Calls to a Blackbox Model

To call a memory service, you must supply a reference to a memory service declaration. This exercise annotates memory references in the blackbox model which in turn generate memory references in the architectural memory hierarchy.

1. In the Behavior diagram you created in the previous exercise, right click the *Adder* and choose *Edit View* from the pop-up menu.

The Edit View dialog opens.



2. Choose *blk_cpp* and click *OK*.

The *blk_cpp* template text window opens.

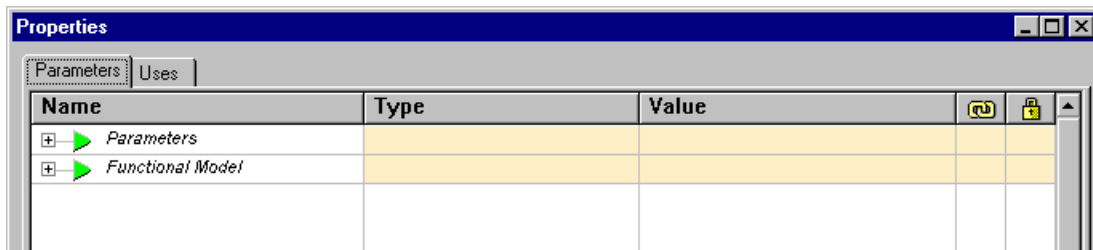
```
add_mult.adder.blk_cpp black.cpp
#include "black.h"
#include <assert.h>
CPP_MODEL_IMPLEMENTATION::CPP_MODEL_IMPLEMENTATION(const ModuleProto &proto, Instance:
: CPP_MODEL_INTERFACE(proto, inst)
{
}
void CPP_MODEL_IMPLEMENTATION::Init()
```

VCC Architecture Services Tutorial

Architecture Services Tutorial

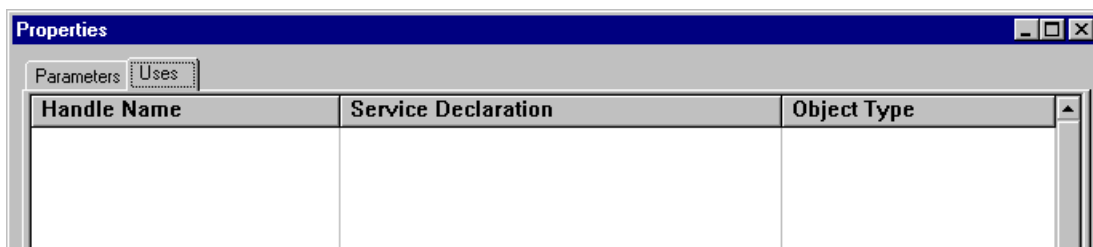
3. Right click in the text window and choose *Properties* from the pop-up menu.

The Properties dialog opens. The dialog defaults to the *Parameters* tab.



4. Click the *Uses* tab.

The *Uses* tab displays only the field titles. There are no defined properties.



5. Right click in the blank area in the dialog box, then choose *New Property* from the pop-up menu.

The plus sign in the Handle Name field indicates that you can enter the name of the property.

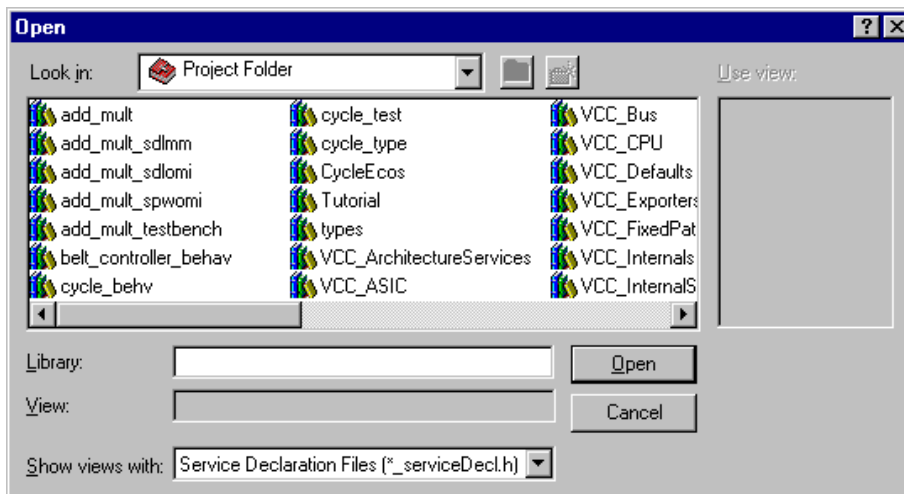
6. Type *meminterface* in the Handle Name field, then tab to the Service Declaration field.

VCC Architecture Services Tutorial

Architecture Services Tutorial

- Click the browser button (...).

The browser opens to your Project Folder.

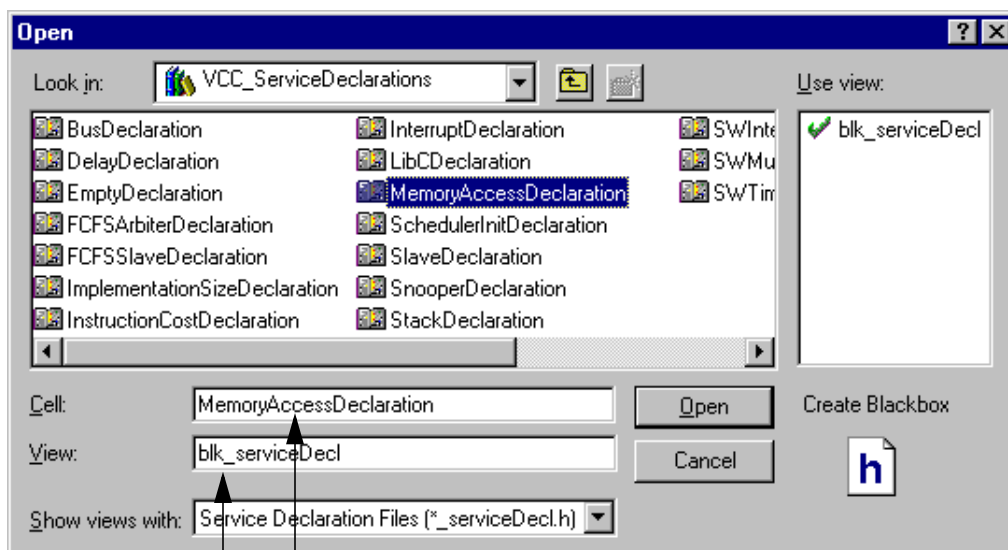


- Double-click the *VCC_ServiceDeclarations* library icon.

- Double-click the *MemoryAccessDeclaration* cell icon.

The view name defaults to *blk_serviceDecl*.

The browser displays the cell and view names you have selected.



Selected Cell name.

Default View name.

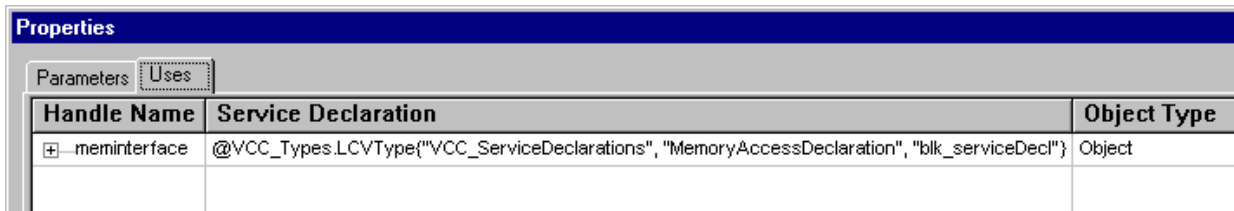
VCC Architecture Services Tutorial

Architecture Services Tutorial

10. Click *Open*.

11. In the Properties dialog, double-click in the Object Type field and enter the object name *Object*.

When you finish, your Properties dialog should look like this:



12. Click *OK* in the Properties dialog.

13. To close the *blk_cpp* template text window, click the X in the upper right corner.

14. To save the mapping diagram, choose *File > Save*.

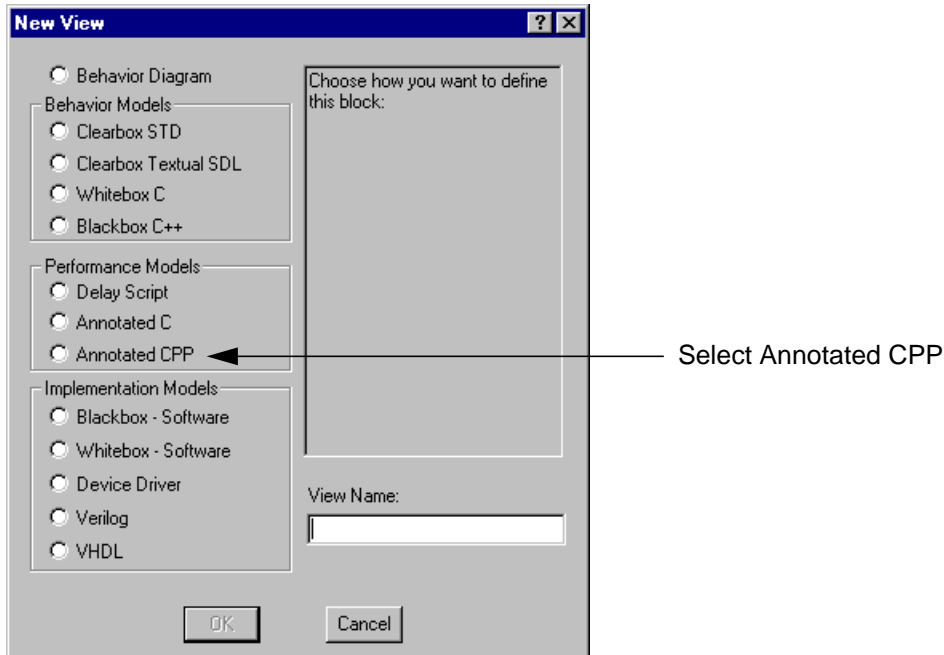
Note: If you are unable to add the service declaration:

- ☐ Close your *Tutorial.mapping_config:mapping* cellview
- ☐ Open *Tutorial_Examples.Example9:mapping*
- ☐ Save the example as *Tutorial.mapping_config:mapping*

Specify the Performance Model

1. In the Behavior diagram, right click the *Adder* and choose *New View* from the pop-up menu.

The New View dialog opens.

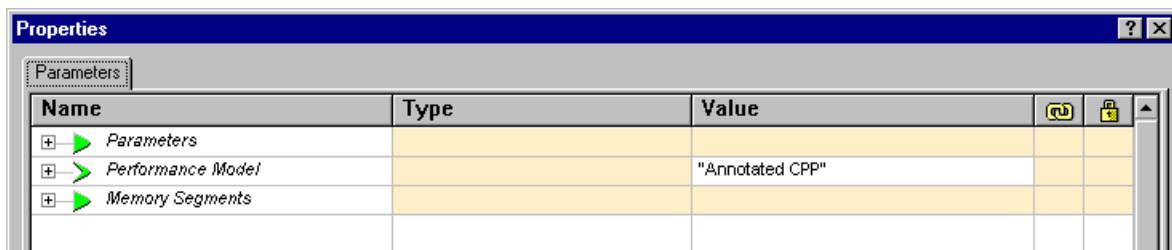


2. Under Performance Models, choose *Annotated CPP*, then click *OK*.

A dialog asks if you want to update the interface of the cell *add_mult.adder*.

3. Click *Yes* in the dialog.

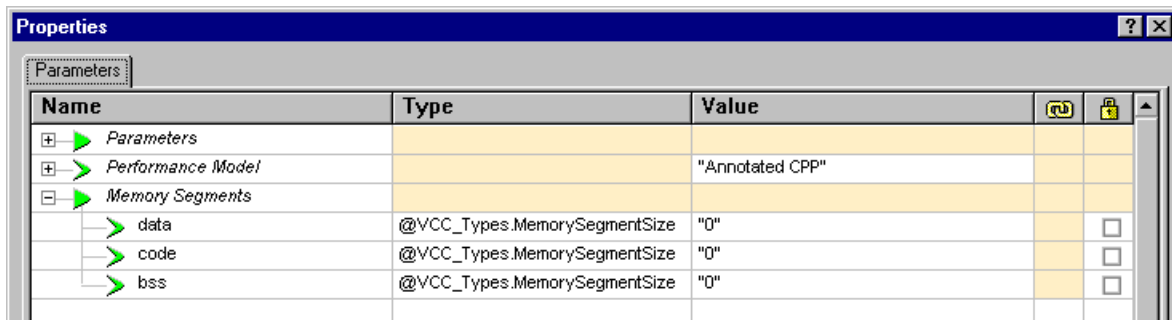
The Properties dialog opens so you can specify properties for the view you just created.



VCC Architecture Services Tutorial

Architecture Services Tutorial

4. Click the plus sign in front of the Memory Segments name.



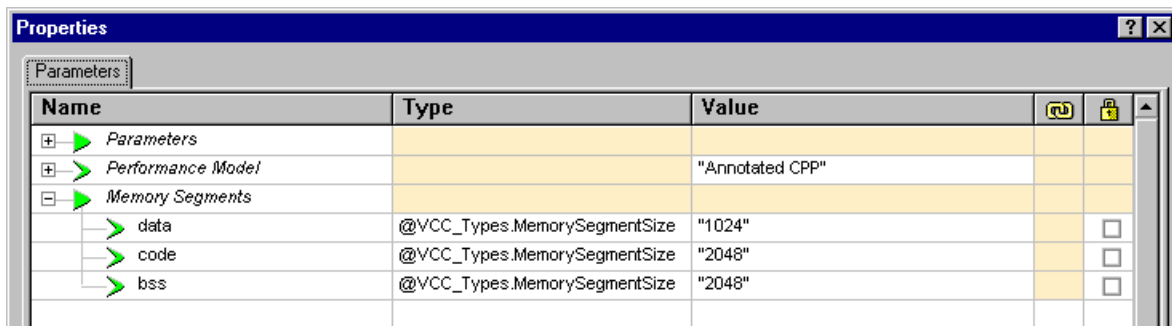
5. Replace the "0" in the Value fields with the following values:

data " 1 0 2 4 "

code " 2 0 4 8 "

bss " 2 0 4 8 "

Note: You must enclose the values with quotations.



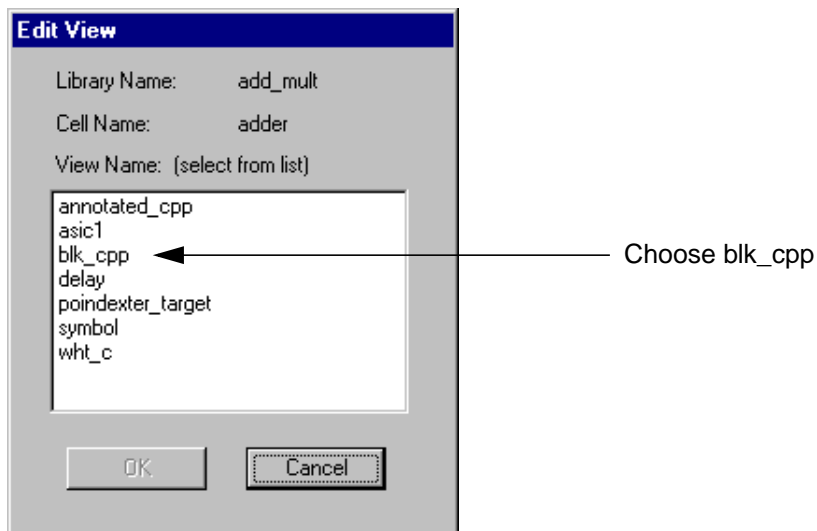
6. When you finish entering all the values, click **OK**.

Embed Delays

If the *add_mult.adder:blk_cpp black.cpp* text window is still open, skip to step 3.

1. In the Behavior diagram, right click the *Adder* and choose *Edit View* from the pop-up menu.

The Edit View dialog opens.



2. Choose *blk_cpp* and click *OK*.

The *blk_cpp* template text window opens.

```
add_mult.adder.blk_cpp black.cpp

#include "black.h"

#include <assert.h>

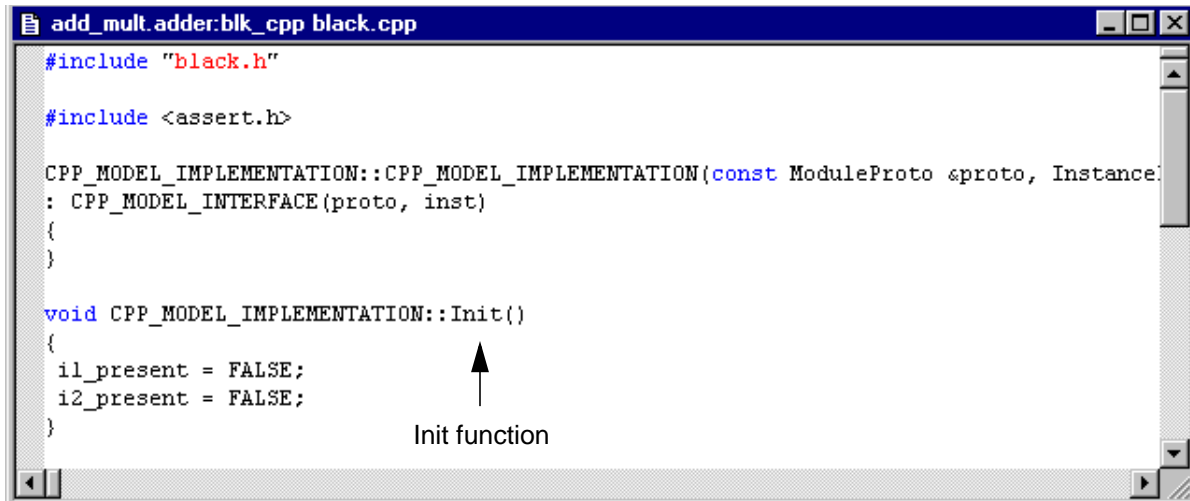
CPP_MODEL_IMPLEMENTATION::CPP_MODEL_IMPLEMENTATION(const ModuleProto &proto, Instance:
: CPP_MODEL_INTERFACE(proto, inst)
{
}

void CPP_MODEL_IMPLEMENTATION::Init()
```

VCC Architecture Services Tutorial

Architecture Services Tutorial

You are going to edit the code of the *Init* function.



```
#include "black.h"

#include <assert.h>

CPP_MODEL_IMPLEMENTATION::CPP_MODEL_IMPLEMENTATION(const ModuleProto &proto, Instance:
: CPP_MODEL_INTERFACE(proto, inst)
{
}

void CPP_MODEL_IMPLEMENTATION::Init()
{
    i1_present = FALSE;
    i2_present = FALSE;
}
```

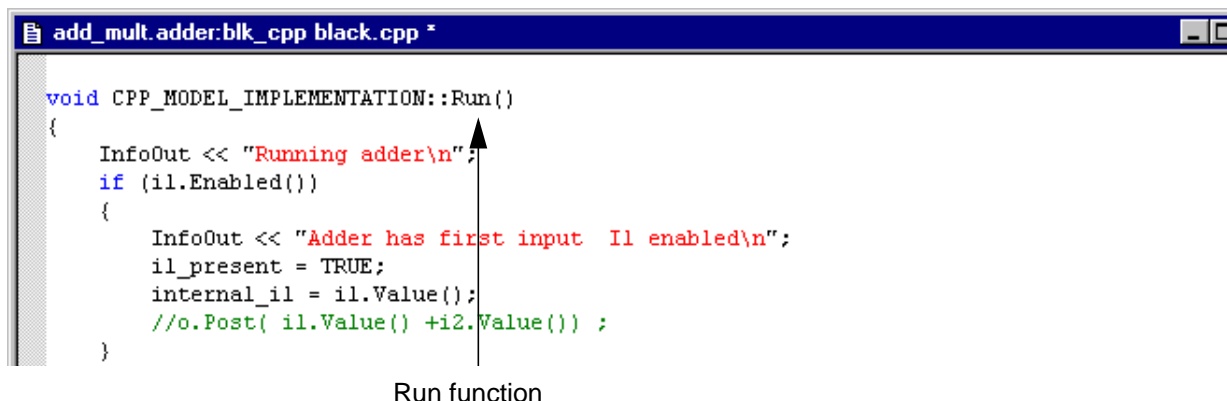
3. Add the following lines to the *Init* function code:

```
void CPP_MODEL_IMPLEMENTATION::Init()
{
    module = meminterface.registerModule(this);
    textSeg = meminterface.registerModuleSegment(module, "code");
    dataSeg = meminterface.registerModuleSegment(module, "data");
    bssSeg = meminterface.registerModuleSegment(module, "bss");
    i1_present = FALSE;
    i2_present = FALSE;
}
```

Add these four lines of code.

VCC automatically changes keywords to red and blue text.

You are now going to embed a delay in the *Run* function.



```
void CPP_MODEL_IMPLEMENTATION::Run()
{
    InfoOut << "Running adder\n";
    if (i1.Enabled())
    {
        InfoOut << "Adder has first input I1 enabled\n";
        i1_present = TRUE;
        internal_i1 = i1.Value();
        //o.Post( i1.Value() +i2.Value() );
    }
}
```

VCC Architecture Services Tutorial

Architecture Services Tutorial

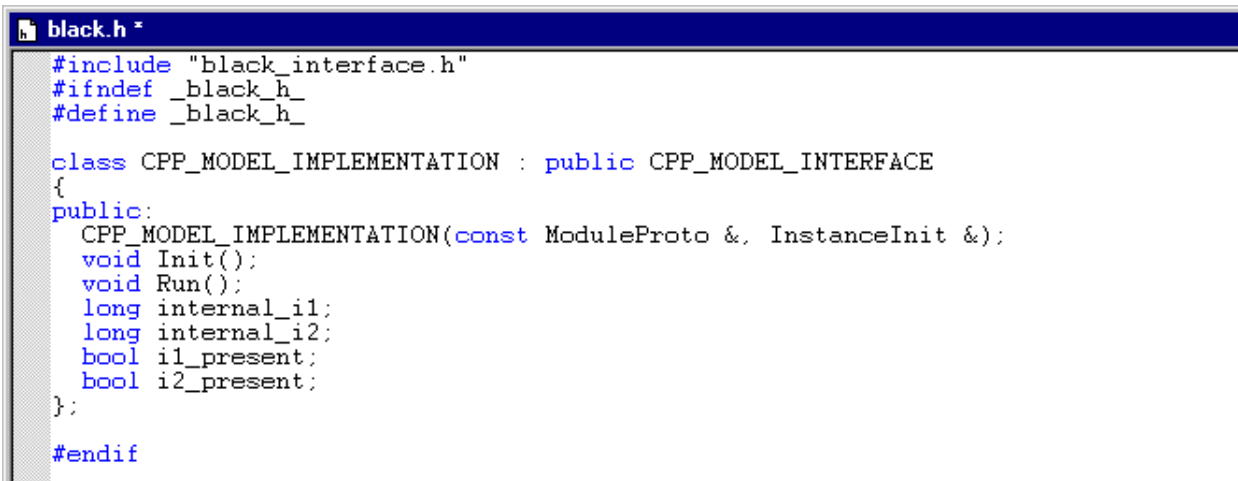
4. Add the following lines of code to the *Run* function code:

```
void CPP_MODEL_IMPLEMENTATION::Run()
{
    meminterface.reference (module, textSeg, 320, meminterface.read);
    meminterface.reference (module, textSeg, 480, meminterface.write);
    meminterface.reference (module, bssSeg, 640, meminterface.write);
}
```

↑
Add these three lines of code.

These three lines of code introduce the following delays:

- ☐ A delay equivalent of memory read for 320 bytes into the text segment.
 - ☐ A delay equivalent of memory write for 480 bytes into the text segment.
 - ☐ A delay equivalent of memory write for 640 bytes into the bss segment.
5. To save the changes, make sure the *add_mult.adder:blk_cpp black.cpp* text window is the active window, and choose *File > Save*.
6. To open the *black.h* file, right click in the *add_mult.adder:blk_cpp black.cpp* text window and choose *Edit black.h*.



```
black.h *
#include "black_interface.h"
#ifndef _black_h_
#define _black_h_

class CPP_MODEL_IMPLEMENTATION : public CPP_MODEL_INTERFACE
{
public:
    CPP_MODEL_IMPLEMENTATION(const ModuleProto &, InstanceInit &);
    void Init();
    void Run();
    long internal_i1;
    long internal_i2;
    bool i1_present;
    bool i2_present;
};

#endif
```

VCC Architecture Services Tutorial

Architecture Services Tutorial

7. Add the following lines of code to the *black.h* file:

```
public:
    CPP_MODEL_IMPLEMENTATION(const ModuleProto &, InstanceInit &);
    void Init();
    void Run();
    long internal_i1;
    long internal_i2;
    bool i1_present;
    bool i2_present;
    long module;
    long textSeg;
    long dataSeg;
    long bssSeg;
};
```

← Add these four lines of code.

8. To save the changes, make sure the *add_mult.adder:blk_cpp black.h* text window is the active window, and choose *File > Save*.
9. To close the text window, click the X in the upper-right corner.
10. To save the *Tutorial.mapping_config:mapping*, choose *File > Save*.
11. To analyze the effect of the delays, follow the steps in “Run a Simulation” and run the simulation.

Note: If you are unable to perform a successful simulation:

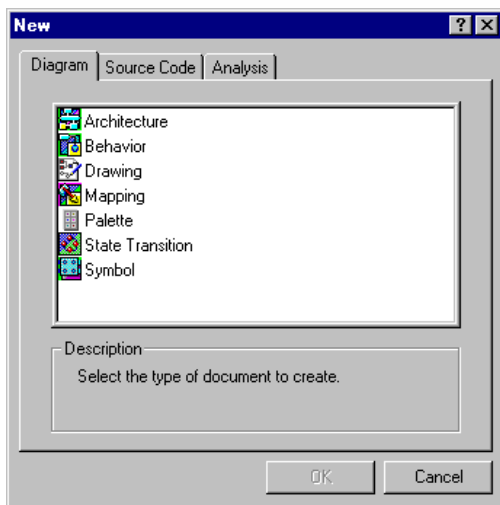
- ☐ Close your *Tutorial.mapping_config:mapping* cellview
- ☐ Open *Tutorial_Examples.Example10:mapping*
- ☐ Save the example as *Tutorial.mapping_config:mapping*
- ☐ Rerun the simulation and analyze the effects of the delays.

Supplemental Exercise: Create a Custom Architecture Model

You can create your own custom architecture model and assign architecture services to it.

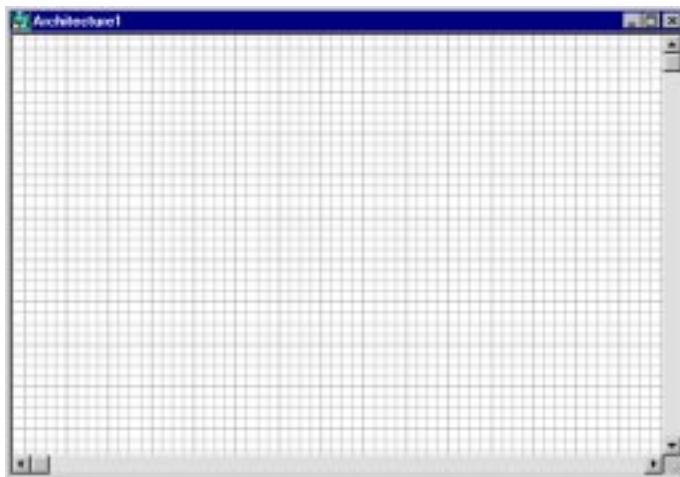
Create a New Block

1. Choose *File > New*.



2. Choose *Architecture* and click *OK*.

A new empty design titled *Architecture1* opens.

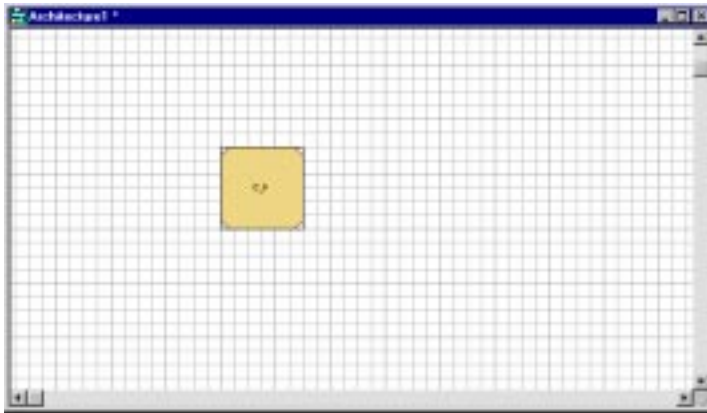


3. Choose *Architecture > Storage Resource*.

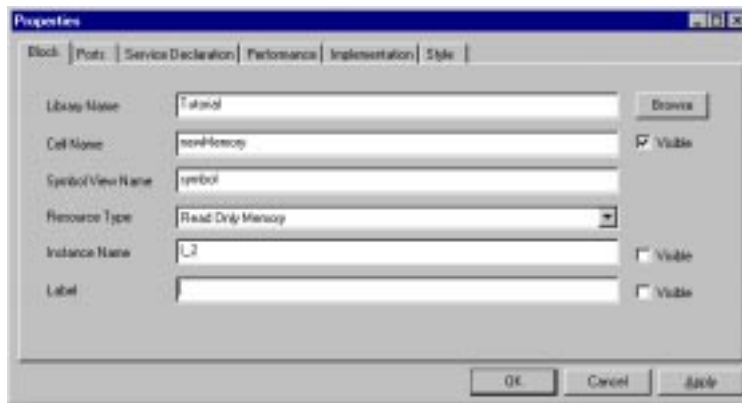
VCC Architecture Services Tutorial

Architecture Services Tutorial

4. Move your cursor into the Architecture diagram and you'll see a block attached to the cursor.
5. Click in the diagram to place the block.



6. Right click on the block and choose *Properties* from the pop-up menu.
The Properties dialog opens with the *Block* tab displayed.



7. Enter the following values in the Properties dialog:

Library Name	Tutorial
Cell Name	newMemory
Symbol	symbol

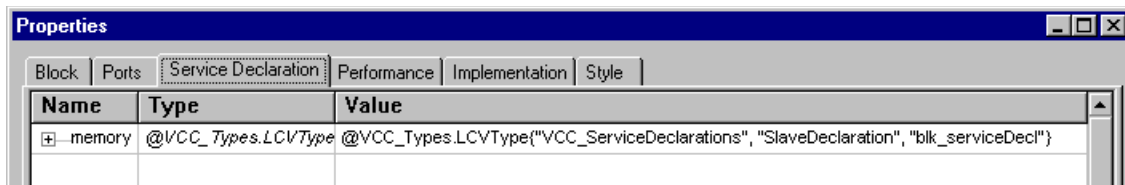
8. In the Resource Type field, click the down arrow and choose *Read Only Memory*.

Do not click *OK* in the Properties dialog. You will continue using this dialog to add service declarations.

Add Service Declarations

1. In the Properties dialog, click the *Service Declaration* tab.

The Properties dialog displays the default service declaration.



2. To add a service declaration, right click in the blank area of the dialog and choose *New Property* from the pop-up menu.

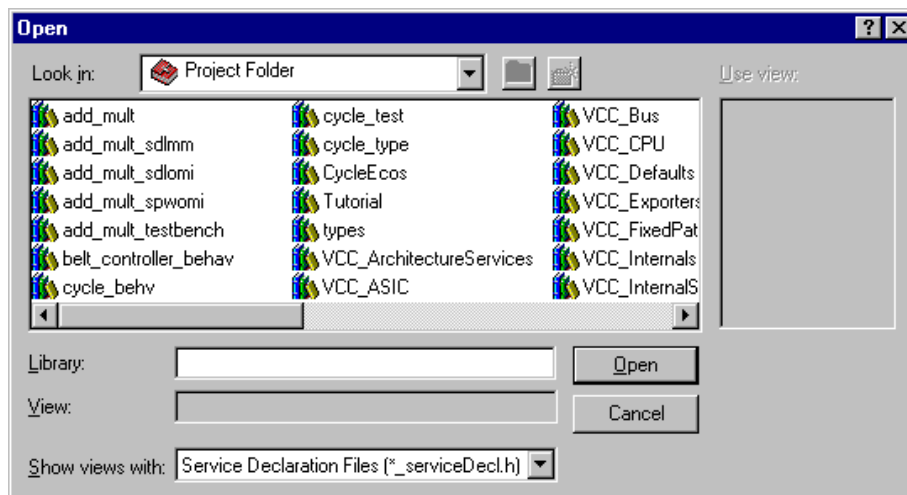
A plus sign in the Name field indicates that you can enter a new service declaration name.

3. Type *newMemory* in the Name field and tab to the Value field.

You cannot edit the Type field for these properties.

4. Click the browser button (...).

The browser opens to your Project Folder.



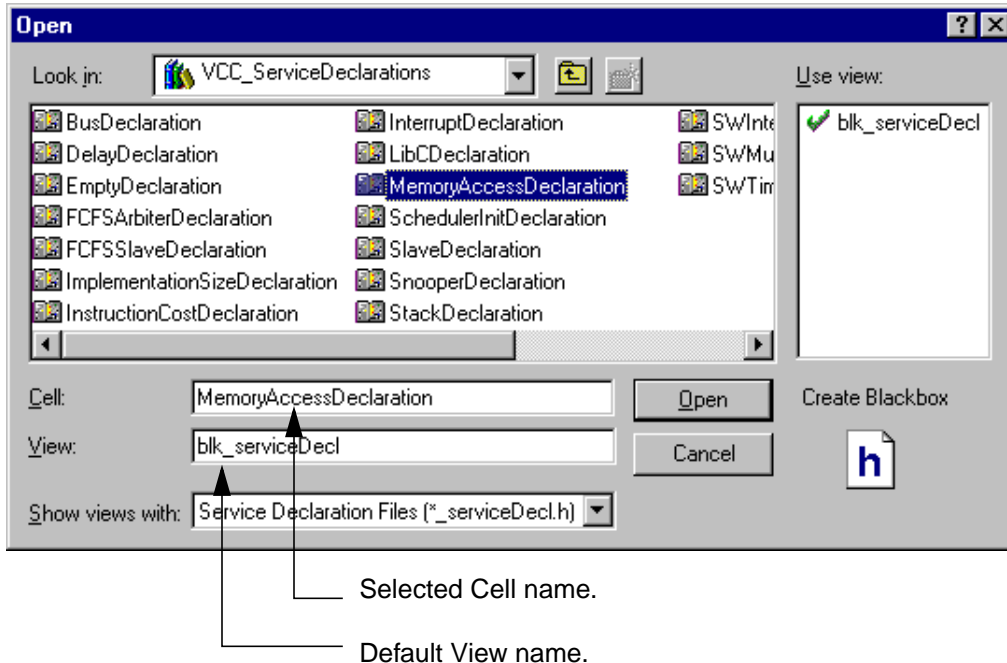
5. Double-click the *VCC_ServiceDeclarations* library icon.
6. Double-click the *MemoryAccessDeclaration* cell icon.

The view name defaults to *blk_serviceDecl*.

VCC Architecture Services Tutorial

Architecture Services Tutorial

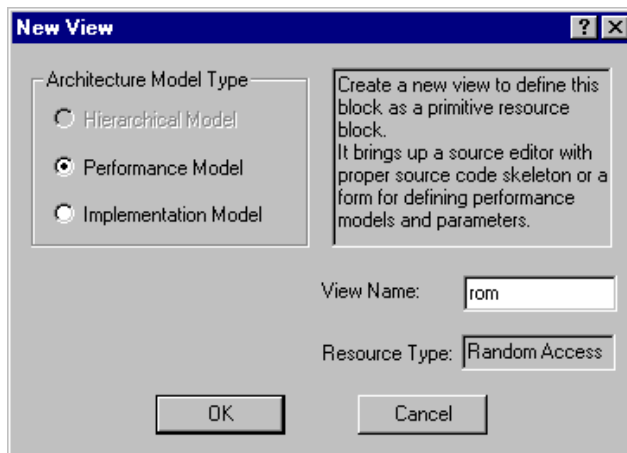
The browser displays the cell and view names you have selected.



7. Click *Open*.

Bind the View

1. Right click on the block and choose *New View* from the pop-up menu.



2. In the View Name field, enter *rom* and click *OK*.

The Properties dialog opens.

VCC Architecture Services Tutorial

Architecture Services Tutorial

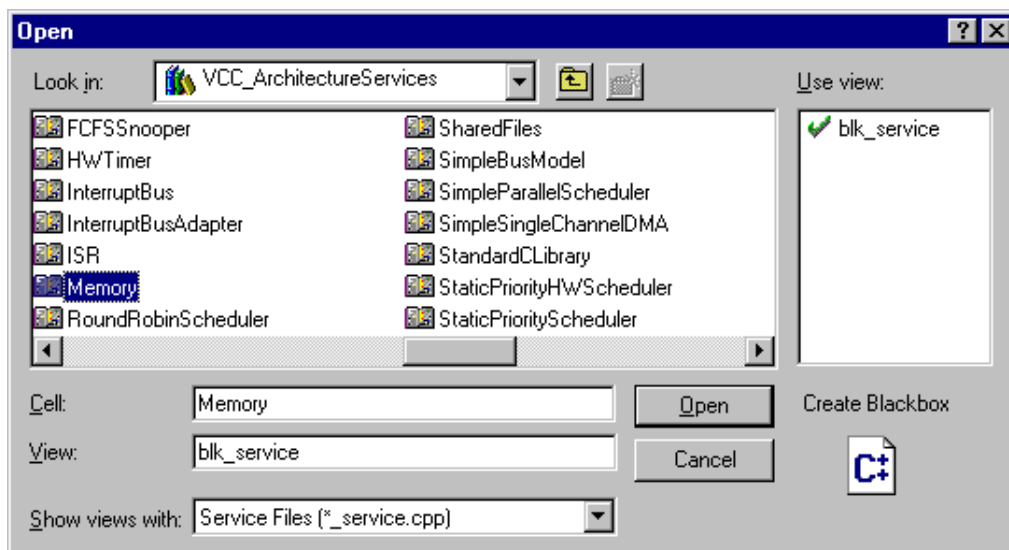
3. Click the *Service Binding* tab.
4. Double-click in the Value field for the *memory* service so the browse button appears, then click the browse button.

The browser opens to your Project Folder.

5. Double-click the *VCC_ArchitectureServices* library icon.
6. Double-click the *Memory* cell icon.

The view name defaults to *blk_service*.

The browser displays the cell and view names you have selected.



7. Click *Open*.
8. Click *OK* in the Properties dialog.

Glossary

A

architecture model

Represents a block in an architecture design. Each model represents a specific implementation of the block in hardware or software and has implementations written in Verilog, VHDL, layout description, C, assembly code, or similar languages.

architecture primitive

The lowest-level functional component in a design. Standard primitives are available on the Architecture menu.

B

behavior diagram

A behavior diagram is composed of ports, parameters, instances, and connectivity.

behavior model

A behavior model in the VCC environment defines the system-level functionality used during simulation.

behavior primitive

The lowest-level behavior component in a design. Standard primitives are available on the Behavior menu.

bind a view

When you bind a view to an architecture block, you tell the VCC system which of the available performance views to implement when you simulate the block. The different views can offer differences in accuracy and simulation performance.

C

cellview

The lowest level of the file structure *library.cell:view*. Users create and modify data at the cellview level. When you instantiate an object, you are placing its cellview.

D

diagram (architecture, behavior)

A graphical collection of objects, such as blocks, primitives, symbols, or connections.

diagram (mapping)

A hierarchical design where behavior and architecture models are mapped and simulated.

E

error

Text message that informs you of unacceptable situations in your design when you perform a code generation or simulation. Errors cause the code generation or simulation to fail.

I

implementation (architecture)

Performance model of an architecture component. Reflects a hardware implementation of that component.

implementation (behavior)

Clearbox, whitebox, or blackbox description of a model. You can have several implementations of the same function in a library, each having the same interface, but with a different implementation.

info

Text message that informs you of the progress of a code generation or simulation.

instance name

As you place a block or primitive in a behavior or architecture diagram, VCC assigns each object an instance name in the form *I_0*, *I_1*, and so forth. VCC refers to instances by this identifier in parameters and error messages.

instantiate

To place a copy of a primitive component or library.cell:view in a design. In VCC, this is as simple as choosing a primitive from a menu or clicking on the symbol view in the project folder, dragging the symbol to your design, and dropping the symbol.

VCC Architecture Services Tutorial

Glossary

L

library cell

A path name to identify a cell in the Cadence library structure. Usually it has the form *library.cell*.

In the NT file structure, the path is specified as *C:\MyWorkspace\library\cell*.

library cell view

A library element and path name to identify components and files in the Cadence library structure. It usually has the form *library.cell:view*.

In the NT file structure, the path is specified as *C:\MyWorkspace\library\cell\view*

library

A library is a collection of directories. Each directory contains files such as symbols, interface files, behavior models, or architecture models.

M

model

A functional representation of a device or system that is delivered in object code format.

model instance

A customized version of a model that has been incorporated into a design.

N

NT

Windows NT. The Microsoft operating system used in a networked environment.

P

parameter

A means by which an application or user can customize the behavior or characteristics of a model instance when it is created. A parameter has a constant value.

pattern

An arrangement of services that collectively model a communication path from sender to receiver. Patterns support the modeling of communication between software and hardware, hardware and software, software and software, or hardware and hardware.

VCC Architecture Services Tutorial

Glossary

performance model

A model that estimates time, power, area, cost, and so forth. In the VCC context, a performance model usually refers to a delay model.

port

Objects in an application that let the model and the application interact during simulation. Ports can be Input, Output, or I/O.

port data type

Defines the information or fields required in the port data structure for the model to work.

primitive (C++)

A function or operator built into C++. Primitives typically include the arithmetic and logical operations (such as *plus*, *minus*, *and*, *or*) and are implemented by a small number of machine language instructions.

project folder

A project folder contains the libraries you can use with the VCC software. A project folder is defined by a *cds.lib* file, which is a listing of libraries.

S

scheduler

Represents a scheduling policy for an architecture resource that governs how the behavior models assigned to the resource gain access and share the resource.

service declaration

Service declarations are defined in the *VCC_InternalServices* library, or users can define their own declarations. A service declaration's header file declares a set of functions that define the service in terms of a C++ abstract base class.

service definition

Service definitions implement service declarations. Service definitions are defined in the *VCC_ArchitectureServices* library.

simulation time

A representation of the time your design takes to execute. Simulation time is not the same as elapsed time to run the simulation.

VCC Architecture Services Tutorial

Glossary

V

VCC

Virtual Component Co-Design. An integrated environment for development of hardware and software virtual components.

W

warning

Text message that informs you of unexpected situations in your design when you perform a code generation or simulation. Warnings do not cause the code generation or simulation to fail.

VCC Architecture Services Tutorial

Glossary
